트루타입 폰트에 내장된 한글 비트맵 데이타의 압축

(Hangul Bitmap Data Compression Embedded in TrueType Font)

한 주 현 † 정 근 호 † 최 재 영 ††
(Joohyun Han) (Geunho Jeong) (Jaeyoung Choi)

요 약 최근에 PDA, IMT-2000, e-Book 등이 보편화되면서 이러한 기기들을 사용하는 사용자의 수가 크게 증가하고 있다. 그러나 아직도 사용가능한 메모리의 크기는 데스크톱 컴퓨터에 비해 상당히 적은 편이다. 이런 제품들에서, 트루타입 폰트는 품질 좋은 글꼴을 요구하는 사용자들이 증가함에 따라 수요가 증가하고 있으며, Windows CE를 탑재한 제품들에서는 기본적으로 사용되고 있다. 하지만 트루타입 폰트의 크기는 PDA와 e-Book과 같은 적은 메모리를 가진 제품들의 상당히 많은 공간을 차지하게 된다. 그러므로 트루타입 폰트의 크기를 줄이려는 노력이 요구된다.

본 논문은 트루타입 폰트에 내장된 비트맵 데이타(EBDT) 중에 한글 부분만을 줄이기 위해 2 단계의 압축과정을 거친다. 1 단계에서는 비트맵을 초성, 중성, 종성의 형태로 분리하여 합성 비트맵으로 구성하고, 2 단계에서는 분리된 각각의 비트맵들의 중복을 조사하여 제거하게 된다. 본 논문은 한글 완성형과 조합형 트루타입에 내장된 비트맵을 압축하였으며, 완성형 폰트의 경우 35%, 조합형 폰트의 경우 7%의 압축률을 보인다. 또한 완성형 트루타입의 경우 전체 트루타입 폰트의 9.26%의 압축률을 보인다.

키워드 : 트루타입 폰트, 비트맵, 한글 압축

Abstract As PDA, IMT-2000, and e-Book are developed and popular in these days, the number of users who use these products has been increasing. However, available memory size of these machines is still smaller than that of desktop PCs. In these products, TrueType fonts have been increased in demand because the number of users who want to use good quality fonts has increased, and TrueType fonts are of great use in Windows CE products. However, TrueType fonts take a large portion of available device memory, considering the small memory sizes of mobile devices. Therefore, it is required to reduce the size of TrueType fonts.

In this paper, two-phase compression techniques are presented for the purpose of reducing the size of hangul bitmap data embedded in TrueType fonts. In the first step, each character in bitmap is divided into initial consonant, medial vowel, and final consonant, respectively, then the character is recomposed into the composite bitmap. In the second phase, if any two consonants or vowels are determined to be the same, one of them is removed. The TrueType embedded bitmaps in Hangeul Wanseong (pre-composed) and Hangul Johab (pre-combined) are used in compression. By using our compression techniques, the compression rates of embedded bitmap data for TrueType fonts can be reduced around 35% in Wanseong font, and 7% in Johab font. Consequently, the compression rate of total TrueType Wanseong font is about 9.26%.

Key words : TrueType Font, Bitmap, Hangul Compression

· 본 논문은 2006년도 숭실대학교 교내 연구비의 지원으로 연구되었습니다.
† 학생회원 : 숭실대학교 컴퓨터학과
jhhan@ss.ssu.ac.kr
gnho@ss.ssu.ac.kr
†† 종신회원 : 숭실대학교 컴퓨터학과 교수
choi@ssu.ac.kr
논문접수 : 2005년 9월 13일
심사완료 : 2006년 4월 19일

1. 서 론

현재 개발되고 있는 PDA, IMT-2000, e-Book 등의 시스템에서도 윈도우에서 사용되는 트루타입 폰트 정도의 미려한 글꼴을 요구하면서 트루타입 폰트를 사용하는 제품들이 늘고 있는 추세이다. 윈도우에서 트루타입

폰트의 사용은 메모리의 크기나 가격 면에서 문제가 되지 않는다. 그러나 PDA와 같은 제품은 메모리 크기는 보통 32메가 정도이며, 가격 면에서도 일반 컴퓨터 메모리 가격보다 월등히 비싸다. 보통 트루타입 폰트는 한글과 한자를 포함하는 파일인 경우 크기가 4 MB에서 8 MB 정도를 차지한다. 윈도우나 리눅스 커널 크기가 800 KB인 것과 비교해 보면 상당히 큰 크기이다. 이런 트루타입 폰트를 적은 메모리를 가진 시스템에 사용하는데 무리가 있다[1].

내장된 비트맵 데이타(EBDT, Embedded Bitmap Data)는 트루타입 폰트의 1/4 크기를 차지하며, 주로 시스템에서 사용되는 8에서 11 포인트 크기의 비트맵 데이타를 저장하고 있다. 또한 이 크기의 폰트들은 래스터라이저에 의한 힌팅(hinting) 등의 작업을 거치지 않고 바로 비트맵 데이타를 사용하기 때문에 속도 면에서도 빠른 이점이 있다. 이와 같은 내장된 비트맵 데이타의 크기를 줄임으로써 PDA와 같은 적은 메모리를 사용하는 제품들에서 효과적으로 사용될 수 있으며, 일반 PC 환경에서는 폰트 캐쉬 크기를 절약함으로써 시스템의 속도를 향상시킬 수 있다.

본 논문은 트루타입 폰트에 내장된 한글 비트맵 데이타(EBDT)를 압축함으로써 트루타입 폰트 크기를 줄이는 것을 목적으로 한다. 이를 위해 합성 비트맵 방식을 이용한 완성형 트루타입 비트맵을 조합형과 유사한 형태로 변형함으로써 중복 사용되는 데이타의 양을 줄이는데 초점을 맞추고 있다. 본 논문에서는 두 단계의 압축과정을 통해 트루타입 비트맵의 크기를 줄인다. 1단계에서는 합성 비트맵 방식을 이용하여 완성형 트루타입 비트맵을 조합형 트루타입 비트맵으로 변환한다. 이 과정에서 공백 등 불필요한 부분이 제거된다. 2단계에서는 1단계에서 조합형 트루타입 비트맵으로 변경된 데이타들의 중복성을 조사하여 제거한다. 이 과정에서도 중복된 비트맵 데이타의 수만큼 데이타의 크기가 줄어든다. 이와 같은 압축과정을 통해 기존의 트루타입 폰트의 크기를 줄임으로써 PDA와 같이 적은 메모리를 사용하는 모바일 장비에 활용될 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 트루타입 폰트 구조 및 폰트 압축 관련기술에 대한 비교연구를 설명한다. 3장에서는 트루타입의 내장된 한글 비트맵 데이타를 압축하기 위한 방법을 제안한다. 4장에서는 본 논문에서 제안하는 압축방식을 이용한 실험 결과를 제시하며, 5장에서는 결론 및 향후계획에 대해 논의한다.

2. 관련연구

2.1 트루타입 폰트

트루타입 폰트(TrueType Font)는 애플 컴퓨터 회사와 마이크로소프트사에 의해서 공동으로 개발되었으며, 마이크로소프트사의 윈도우 계열에서 주로 사용된다. 폰트 처리방식으로 볼 때 윤곽선 폰트에 해당한다[2-4]. 트루타입 폰트는 여러 개의 테이블들로 구성된다. 기본적으로 필요한 10개의 테이블과 추가적으로 사용하는 14개의 테이블로 구성된다. 본 논문에서 트루타입 폰트를 압축하기 위해 사용하는 테이블 중에 중요한 테이블로는 cmap(character to glyph mapping), EBLC(Embedded Bitmap Location Data), EBDT(Embedded Bitmap Data) 테이블을 들 수 있다[5]. cmap 테이블은 문자코드에 해당하는 글립 인덱스를 찾는데 사용되는 테이블이다. EBLC 테이블은 글립 인덱스에 해당하는 비트맵의 위치 정보를 찾기 위해 사용되고, EBDT 테이블은 실제로 비트맵 데이타를 저장하고 있는 테이블로 비트맵의 위치정보를 통하여 접근할 수 있다[6].

2.1.1 cmap, EBLC, EBDT 관계

특정 문자코드에 해당하는 트루타입 비트맵 데이타를 찾기 위해서는 cmap, EBDT, EBLC 테이블을 이용해야 한다. 예를 들어 그림 1에서 보듯이, g라는 문자코드(0xFF47)에 해당하는 비트맵 데이타를 구하는 과정은 g에 해당하는 문자코드 0xFF47이 cmap 테이블을 이용해서 글립 인덱스를 구하게 되고, 구해진 인덱스 값을 EBLC 테이블에 적용하여 비트맵 위치 값을 얻게 된다. 얻어진 위치 값에 해당하는 데이타를 EBDT 테이블을 통하여 찾게 되면 g라는 비트맵 데이타를 얻을 수 있다[7].

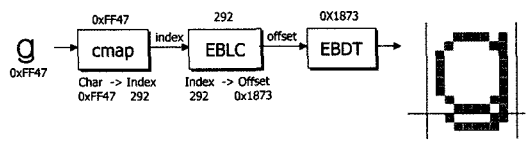

그림 1 cmap, EBLC, EBDT 관계도

2.1.2 합성 비트맵

합성 비트맵(composite bitmap)은 데이타를 직접 저장하지 않고 다른 비트맵을 가리키고 있는 포인터로서 비트맵을 표현하는 방식이다. 그림 2에서 보면 합성 비트맵 '가'는 글립 비트맵 1256번과 1976번에 대한 옵셋(offset) 정보를 통하여 표현될 수 있다. 그렇기 때문에 비트맵 데이타를 직접 가지고 있는 경우에 비해 저장공간을 효율적으로 사용할 수 있는 장점이 있다.

2.2 폰트 압축 관련기술

일반적으로 비트맵 데이타의 압축을 위해서는 RLE(Run Length Encoding) 방식을 응용한 LZW(Lempel-Ziv-Welch) 알고리즘을 사용하지만, 트루타입 표준에서 이를 지원하지 않기 때문에 내장된 비트맵 데이타를 이와 같은 방식으로 압축할 경우 기존의 트루타입 래스터

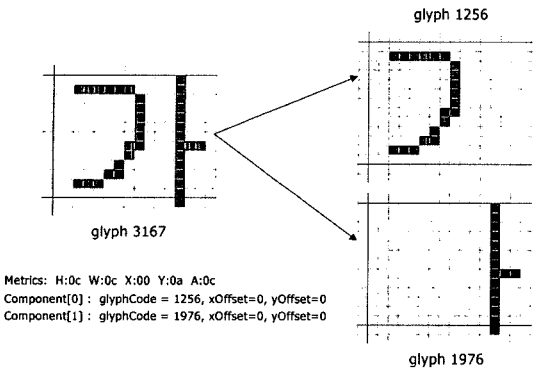

그림 2 합성 비트맵

라이저에서 이를 처리할 수 없고, 래스터라이저를 모두 수정해야하는 문제점이 생기게 된다[8]. 한편 Windows CE.net 이상에서는 아그파의 기술이 적용된 트루타입 오픈을 채택하여, 크기에 문제가 있는 동양권 폰트들을 주로 함께 사용되는 데이타 테이블들로 세분화하여 압축한 형태로 보관하고 있다가, 필요하면 일부분씩 압축을 풀어서 메모리로 탑재하는 기술을 사용하기도 한다. 이 기술은 데이타의 압축해제로 인한 오버헤드가 발생하며, 이를 지원하지 않는 이전의 래스터라이저와의 하위 호환성 문제가 발생한다[9,10].

본 논문에서는 트루타입 표준인 합성 비트맵 방식을 이용하여 비트맵 데이타를 압축하기 때문에 기존의 래스터라이저의 수정없이 트루타입을 지원하는 모바일 기기나 e-Book에서 압축된 트루타입 폰트를 사용할 수 있다. 또한 기존의 완성형 비트맵을 합성 비트맵으로 변경하여 압축할 때 생기는 오버헤드보다 압축으로 인한 트루타입 폰트의 크기 감소율이 더 크기 때문에 다른 폰트 압축 기술에 비해 효율성이 높다.

3. 트루타입 비트맵 데이타의 압축

한글의 초·중·종성 글립을 가지고 있는 조합형 폰트의 경우 대체로 보기 좋은 글자꼴을 얻기 위해서는 초성 10벌, 중성 4벌, 종성 4벌, 또는 초성 8벌, 중성 4벌, 종성 4벌 등으로 구성하는데, 각각 모두 382개나 344개의 자소를 디자인해야 한다. 실제 명조체는 900여 개의 자소가 있어야 이를 조합하여 제대로 서체를 표현할 수 있으므로 기존의 조합형 폰트의 품질이 크게 떨어진다는 것을 알 수 있다[11]. 예를 들면 휴먼 명조체의 경우 초성의 'ㄱ'은 26벌, 'ㅏ'는 5벌이 필요하며, 전체 자소는 909개의 자소가 필요하다. 하지만 조합형의 특성상 각 초성 중성 종성에 대한 정해진 조합 틀을 가지고 있으므로, 벌수를 늘린다고 하더라도 조합 형태에 따라 자소의 모양도 변하는 한글을 제대로 표현하는 폰트를 만들기는 어렵다[12].

조합형 폰트를 완성형 폰트와 비교해 보면 폰트를 만드는데 필요한 시간과 노력, 그리고 폰트 저장에 필요한 공간은 절약되지만 글자의 품질은 완성형보다 뒤떨어진다. 반대로 완성형 폰트는 글자의 품질은 우수하지만 폰트를 만드는데 소요되는 시간과 노력 그리고 저장 공간이 많이 필요하다[13]. 특히 한글 완성형 폰트는 폰트의 중복성이란 단점이 있다. 폰트의 중복성이란 자소가 모여서 한 음절을 이루는 한글의 특성상 완성형 폰트 하나의 음절을 도안하고 저장하면서 불필요하게 동일한 자소를 다시 디자인하고 이를 중복하여 저장하는 것을 말한다. 이러한 폰트의 중복성은 완성형 폰트 저장에 많은 공간을 필요로한다[14]. 만약 이들 중복된 자소를 최소화하여 완성형 폰트를 구성할 수 있다면, 완성형 폰트의 품질을 최대한 유지하면서 조합형 폰트와 유사하게 폰트 생성에 필요한 시간과 노력뿐만 아니라 저장 공간을 모두 절약할 수 있다.

따라서 완성형 폰트에서 중복된 자소들을 뽑아내고 그 자소를 단순 글립으로 구성한 후, 중복된 자소를 사용하는 글립이 단순 글립들을 참조하도록 완성형 폰트를 구성한다. 그러면 완성형 폰트의 품질을 그대로 유지하면서 폰트의 크기는 단순 글립을 저장하기 위한 기억장소만 소모하므로 기억장소는 조합형과 근사하게 된다. 본 논문은 트루타입 폰트에 내장된 한글 비트맵 데이타(EBDT)를 압축하기 위해 두 단계의 처리과정을 거친다. 1 단계에서는 비트맵 데이타를 8 방향 탐색 알고리즘을 이용해서 초성, 중성, 종성의 형태로 재구성하여 합성 비트맵으로 만든다. 2 단계에서는 새로 생성된 비트맵 데이타 중에서 같은 형태의 비트맵을 XOR 연산을 통해 제거한다. 그리고 제거된 비트맵을 가리키는 모든 합성 비트맵은 대표 비트맵을 가리키도록 수정한다.

3.1 비트맵 데이타 추출방식

합성 비트맵을 구성하기 위해서는 비트맵 데이타를 초성, 중성, 종성으로 분리해야 한다. 비트맵 데이타는 0과 1로 표현되기 때문에 그림 3은 비트맵 데이타를 0, 1을 갖는 2차원 배열로 표현하였다. 비트맵 데이타 중에서 1인 부분만을 추출하기 위해서는 8 방향에 대해서 1 값을 갖는지를 확인해야 한다. 왜냐하면 4 방향만을 생각할 경우, 대각선에 있는 1 값을 놓칠 수가 있다. 8 방향 탐색을 위한 탐색 순서는 오른쪽부터 시계 방향으로 진행된다. 좌표축은 x={1,1,0,-1,-1,-1,0,1}, y={0,1,1,1,0,-1,-1,-1}을 이용하고, 먼저 배열의 값이 1인 점을 찾게 되면 그 점을 0으로 바꾼 다음에, 새로운 배열을 생성하여 1 값을 저장한다.

이와 같은 방식을 순환적으로 실행하여 처음 1 값을 찾은 위치로 돌아오면 다시 탐색을 시작하게 된다. 이렇

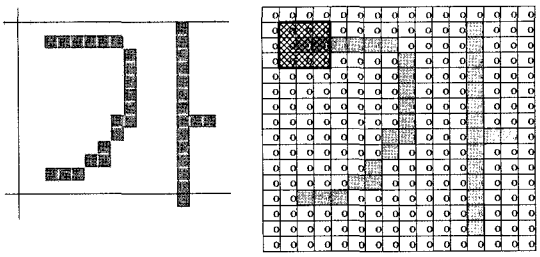

그림 3 추출 전 비트맵 데이타

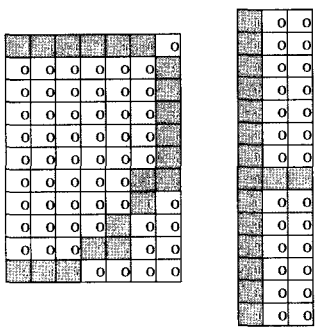

그림 4 추출 후 생성된 비트맵 데이타

게 구한 각각의 비트맵들은 그림 4와 같이 새로운 글립 인덱스를 부여받고 저장된다. 그리고 이들을 가리키는 새로운 합성 비트맵을 생성하게 된다.

3.2 1 단계 압축과정

1 단계 압축과정은 하나의 완성형 비트맵의 초성, 중성, 종성을 각각의 비트맵 데이타로 저장하고 각각에 대한 포인터 값만을 가지는 합성 비트맵으로 변형한다. 1 단계 비트맵 압축과정은 그림 5와 같다. 우선 완성형 비트맵에 해당하는 단순 비트맵(simple bitmap)을 3.1절의 비트맵 데이타 추출방식을 이용하여 글립 단위(초/중/종성)로 추출하여 저장하고, 각각의 글립들을 좌측 상단으로 위치시켜 글립의 크기를 최소화한다. 이렇게 생성된 글립들을 가리키는 새로운 합성 비트맵을 생성하고 이전의 단순 비트맵은 제거한다.

예를 들어, 그림 6에서 글립 20362는 '가'라는 데이타를 가지고 있는 비트맵인데 이를 'ㄱ'과 'ㅏ'로 나누어 각각을 왼쪽 상단으로 이동시킨 후에 저장하고, 이전 비트맵인 '가'는 제거시킨다. 이때 글립 20362를 가리키는 합성 비트맵들이 있는 경우는 그 합성 비트맵의 Component 배열에서 글립 20362에 해당하는 glyphCode 부분을 새로 생성된 글립 30000로 대치시키고, 글립 30001을 추가시켜 이를 가리키게 한 후에 이동한 좌표값을 저장한다. 만일 글립 20362를 가리키는 합성 비트맵이 없는 경우는 글립번호 20362를 가지는 합성 비트맵을 새로 생성하여 글립 30000과 글립 30001을 각각 가리키도록 하고 이동한 좌표 값을 저장하게 된다. 이를 통하여 '가'라는 문자코드는 'ㄱ'과 'ㅏ'를 이용하여 표현하게 된다.

3.3 2 단계 압축과정

2 단계의 압축과정은 1 단계에서 저장된 각각의 비트맵 중에서 동일한 모양을 가진 비트맵을 제거시키고 이를 가리키던 합성 비트맵의 포인터를 수정하는 방식이다. 다시 말해, 합성 비트맵에 의해 참조되는 각각의 글립들은 자소별로 중복성을 조사하고, 중복 글립들 중에 하나의 대표 글립만을 제외한 모든 글립들은 제거된다. 또한 제거된 글립을 참조하는 모든 합성 비트맵들은 대표 글립을 가리키도록 수정한다. 2 단계의 비트맵 압축과정은 그림 7과 같다.

예를 들어, 'ㄱ'에 해당하는 비트맵 데이타는 여러 개 존재하게 되고, 이를 가리키는 합성 비트맵들도 존재하게 된다. 그러나 각각의 'ㄱ'들은 같은 모양일 수 있다. 왜냐하면 비트맵의 좌표축이 한 곳으로 이동되었기 때문에 이전에 같은 모양이면서 위치가 다른 비트맵들도 이제는 같은 모양의 비트맵이 되기 때문이다. 이를 판별하기 위한 방법으로는 두개의 비트맵 데이타를 XOR 연

```
CompressionProgress#1() :
  LOAD a binary metrics of a simple bitmap for compression
  LOOP a number of glyphs in the simple bitmap :
    EXTRACT a glyph in the simple bitmap recursively
            using a direction metrics of X={1,1,0,-1,-1,-1,0,1}, Y={0,1,1,1,0,-1,-1,-1}
    CREATE a binary metrics to save the extracted glyph
    OPTIMIZE a size of the new binary metrics by shifting the glyph
            into the top-left corner
  CREATE a composite bitmap by referencing each glyph
      for substituting the simple bitmap
  DELETE the simple bitmap
```

그림 5 1단계 비트맵 압축 알고리즘

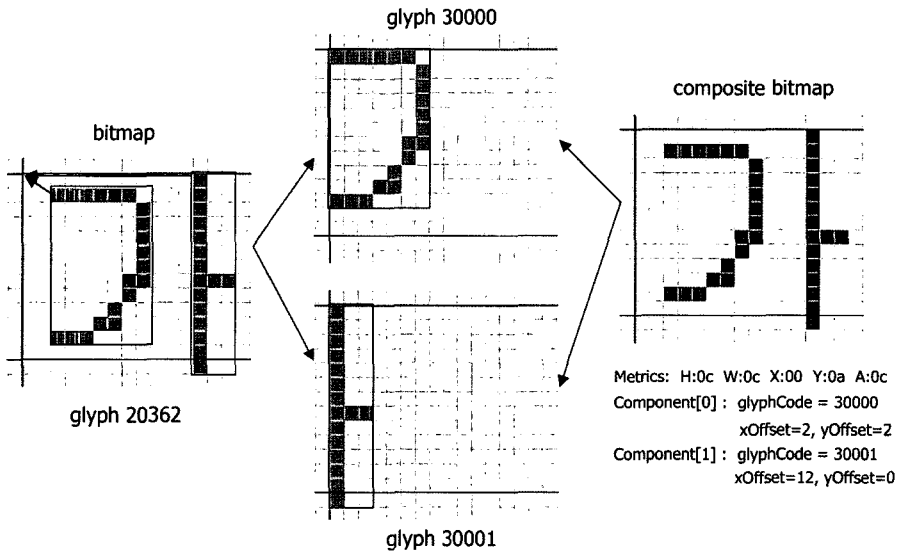

그림 6 트루타입 비트맵 1 단계 압축과정

```
CompressionProgress#2() :
  LOOP a number of glyphs referred by each composite bitmap :
    COMPARE a current glyph with the others
            to find the duplicated glyphs in the same shape
    CASE find the duplicated glyphs :
      MODIFY the pointer of the composite bitmap referring
             the duplicated glyphs into the current glyph
      DELETE the duplicate glyphs
```

그림 7 2단계 비트맵 압축 알고리즘

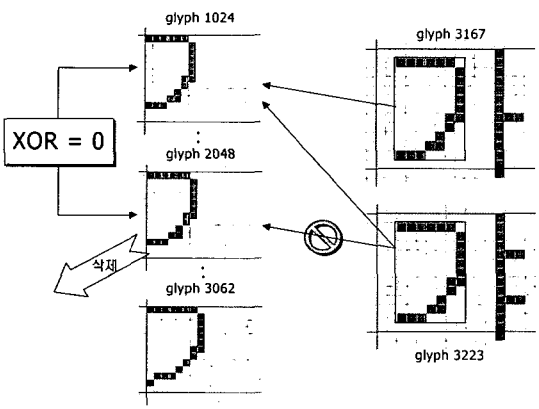

그림 8 트루타입 비트맵 2 단계 압축과정

산에 의해 값이 0이 나오게 되면 같은 것으로 간주한다. 이렇게 찾아낸 같은 모양의 'ㄱ'들을 그림 8에서 보듯이 합성 비트맵으로 표현하기 위해서는, 단 하나의 비트맵 데이타만 필요하기 때문에 하나의 비트맵만을 남겨두고 나머지 비트맵 데이타들은 삭제하게 된다. 그리고 삭제된 데이타를 가리키고 있던 합성 비트맵 포인터는 저장된 비트맵을 가리키도록 재설정한다.

4. 구현 결과

본 논문에서는 트루타입 폰트에 내장된 한글 비트맵 데이타(EBDT)를 압축하기 위한 기법으로 두 가지 압축 과정을 사용하였다. 첫 번째 과정은 매트릭스 정보와 비트맵 데이타를 저장하고 있는 비트맵을 합성 비트맵 방식을 이용하여 각각에 대한 포인터 값만을 가지는 비트맵으로 표현하는 방법이다. 다시 말해서 초성, 중성, 종성을 각각의 비트맵 데이타로 저장한다. 두 번째는 이렇게 저장된 각각의 비트맵 중에서 동일한 모양을 가진 비트맵을 제거시키고 이를 가리키던 합성 비트맵의 포인터를 수정하는 방식이다. 본 논문에서는 완성형과 조합형, 두 종류의 트루타입 폰트를 대상으로 실험하였다. 각각의 파일 특징은 표 1과 같다.

완성형 트루타입 폰트는 특수문자, 영문, 한글 자모를

표 1 완성형과 조합형 트루타입 폰트 비교

	글꼴 이름	전체 크기	EBDT 크기	비중(%)	글꼴 크기
완성형	굴림 옛한글 자모	812 KB	215 KB	26.5 %	12 x 12, 14 x 14, 16 x 16, 24 x 24
조합형	굴림	4034 KB	772 KB	19.1 %	11 x 11, 12 x 12, 13 x 13, 15 x 15

포함하고 있으며, 조합형 폰트는 특수문자, 영문, 한글, 한자를 포함한다. 완성형 트루타입 비트맵인 경우는 합성 비트맵 형태가 존재하지 않으며, 조합형 트루타입 비트맵은 합성 비트맵으로 구성된다.

4.1 1 단계 압축 계산방식

1 단계에서는 비트맵 데이타를 8 방향 탐색 알고리즘을 이용해서 초성, 중성, 종성의 형태로 재구성하여 합성 비트맵을 만든다. 예를 들어, 그림 3의 16 × 16 크기의 완성형 비트맵을 합성 비트맵으로 변환할 경우의 압축률 계산방식은 표 2와 같다.

위의 계산방식 중에 ④번 항목은 조합형일 경우 해당 비트맵을 가리키는 합성 비트맵의 유무에 따라 달라지며, 가리키는 비트맵의 수가 증가함에 따라 4 바이트씩 증가한다. 그리고 압축률 0%의 의미는 비트맵이 나뉘어지면서 줄어드는 크기와 합성 비트맵이 생성되면서 발생하는 오버헤드가 같아 압축이 일어나지 않는 경우를 말한다.

4.2 1 단계 압축 결과

1 단계 압축과정에서는 합성 비트맵 방식을 이용하여 완성형 트루타입 비트맵을 조합형 트루타입 비트맵으로 변환한다. 이때 완성형 트루타입 비트맵에서 글자를 나타내는 부분만 저장되고 공백으로 존재하던 부분은 조합형 비트맵으로 변환되는 과정에서 제거되므로 전체적으로 비트맵 데이타의 크기가 줄어들며, 비트맵의 크기가 클수록 공백부분이 더 많기 때문에 더 좋은 압축률을 보이게 된다. 표 3과 표 4는 완성형과 조합형 트루타입 폰트에 내장된 비트맵 데이타(EBDT) 중에서 한글 부분만을 본 논문의 1 단계 압축과정에 적용한 결과이다.

표 3과 표 4의 결과를 보면, 1단계 압축과정에서 완성형 트루타입 폰트인 경우는 24%의 압축률을 보이는 반면, 조합형 트루타입 폰트는 5%의 압축률을 나타낸다. 이는 완성형 트루타입 폰트는 합성 비트맵 형태가 아니기 때문에 1 단계 과정에서 비트맵이 합성 비트맵으로 변환되는 과정에서 크기가 감소하지만, 조합형인 경우는 이미 비트맵이 합성 비트맵으로 변환되어 있기 때문에 1 단계 압축과정을 거치더라도 크기가 많이 감소하지 않는다. 하지만 조합형 트루타입 비트맵의 경우에서도 비트맵 데이타만의 압축률은 전체 압축률(5%)에 비해 31% 정도로 높다. 그러나 전체 크기 중에 비트맵 데이타가 차지하는 비중이 적고 합성 비트맵이 거의 대부분을 차지하기 때문에 비트맵 데이타만의 압축률로는 전체 크기에 영향을 미치지 못하게 된다.

표 2 1 단계의 압축률 계산방식

	글립(Glyph) 기호	비트맵 데이타의 크기 (Byte)
완성형 비트맵 (압축 전)	① 16 x 16 크기의 '가'	32
합성 비트맵 (압축 후)	② 추출된 'ㄱ'	7 (width) * 11 (height) = 77 = 10
	③ 추출된 'ㅏ'	14 * 3 = 42 = 6
	④ 'ㄱ'과 'ㅏ'를 통해 만들어지는 합성 비트맵 '가'	12 (format 8의 경우) + 4 (1개의 컴포넌트) = 16
	⑤ 압축을 통해 새로 생성된 '가'	10 + 6 + 16 = 32
압축률 (%)	100 - (32 / 32) * 100 = 0 %	

표 3 완성형 트루타입 폰트의 1 단계 압축결과

	12 × 12	14 × 14	16 × 16	24 × 24	결 과
압축 전 (byte)	13,824	19,200	24,576	55,296	112,896
압축 후 (byte)	13,824	19,764	21,956	29,802	85,346
압축률 (%)	0 %	- 3 % (*Overhead*)	11 %	46 %	24 %

표 4 조합형 트루타입 폰트의 1 단계 압축결과

	11 × 11	12 × 12 / 13 × 13	15 × 15	결 과
압축 전 (byte)	x	247,026	256,750	503,776
압축 후 (byte)	x	238,322	242,268	480,590
압축률 (%)	x	4 %	6 %	5 %

4.3 2 단계 압축 계산방식

2 단계에서는 1 단계에서 새로 생성된 비트맵 데이타 중에서 중복 비트맵 데이타를 XOR 연산을 통해 제거한다. 그리고 제거된 비트맵을 가리키는 모든 합성 비트맵을 대표 비트맵을 가리키도록 수정한다. 2 단계의 압축률 계산방식은 표 5와 같다.

2 단계 압축에서는 중복되는 비트맵이 많아질수록 압축률이 증가하며, 합성 비트맵의 포인터를 수정하면서 생기는 오버헤드는 발생하지 않는다.

4.4 2 단계 압축 결과

2 단계 압축에서는 1 단계에서 생성된 비트맵 데이타의 중복을 검사하여 제거하는 과정이다. 중복되는 한글 자소가 많을수록 압축률이 좋아진다. 표 6과 표 7은 2 단계 압축 과정을 거친 완성형과 조합형 트루타입 폰트의 압축률을 나타낸다.

표 6에서 보면, 2 단계 압축과정에서는 완성형 트루타입 폰트의 경우 1 단계의 압축률인 24%에 비해 11%가 감소한 35%의 압축률을 보이며, 이는 전체 크기로 볼 때 9.26%의 압축률을 나타낸다. 또한 표 7의 조합형 트루타입 폰트의 경우도 미소하지만 2%의 감소율로 7%의 압축률을 보이며, 전체 크기의 1.33%의 압축률을 나타낸다. 이와 같은 압축 결과는 트루타입 폰트의 글자 크기가 크면 클수록 압축률이 증가한다. 왜냐하면, 글자가 크면 1 단계 압축과정에서 글자를 제외한 여백부분이 상대적으로 많이 제거되며, 2 단계 압축과정에서도 중복 글립이 차지하는 용량이 상대적으로 크기 때문에 중복 글립이 제거될 때 상대적으로 높은 압축률을 보이기 때문이다. 또한, 조합형 트루타입 폰트가 완성형 트루타입에 비해 1, 2 단계 압축과정에서 적은 감소율을 보이는 이유는 합성 비트맵의 사용과 조합형 트루타입 폰트 제작과정에서 중복을 제거했기 때문이다.

5. 결론 및 향후계획

본 논문에서는 트루타입 폰트에 내장된 한글 비트맵 데이타(EBDT)를 압축하기 위한 기법으로 두 단계의 압축과정을 사용하였다. 첫 번째 과정은 매트릭스 정보와 비트맵 데이타를 저장하고 있는 비트맵을 합성 비트맵 방식을 이용하여 각각에 대한 포인터 값만을 가지는 비트맵으로 표현하는 방법이며, 두 번째는 이렇게 저장된 각각의 비트맵 중에서 동일한 모양을 가진 비트맵을 제거시키고 이를 가리키던 합성 비트맵의 포인터를 수정하는 방식이다. 트루타입 비트맵 압축은 완성형일 경우, 비트맵을 합성 비트맵으로 변환하는 방식보다는 변환되어 저장된 비트맵들의 중복을 제거하는 경우에 더 많이 줄어드는 것을 확인할 수 있고, 조합형인 경우는 완성형의 경우와는 달리 비트맵을 합성 비트맵으로 변환하는 과정이 중복을 제거하는 방법보다는 더 줄어든다.

트루타입 폰트의 압축률은 폰트가 조합형일 때보다는 완성형 비트맵일 때 좋은 압축률을 기대할 수 있다. 왜냐하면 조합형인 경우는 이미 비트맵 데이타들이 초성, 중성, 종성으로 나뉘어져 합성 비트맵으로 구성이 되어 있기 때문에, 각각의 비트맵 데이타를 더 작은 단위로 나누게 되면 참조되는 횟수도 감소할 뿐만 아니라 그 비트맵을 가리키는 합성 비트맵의 수가 많을수록 추가

표 5 2 단계의 압축률 계산방식

	압축 전	압축 후
비트맵 데이타 'ㄱ' (10 Bytes)의 중복 비트맵 개수	2 개	1 개
비트맵 데이타의 크기 (Byte)	20	10
압축률 (%)	100 - (10 / 20) × 100 = 50 %	
비트맵 데이타 'ㄱ' (10 Bytes)의 중복 비트맵 개수	3 개	1 개
비트맵 데이타의 크기 (Byte)	30	10
압축률 (%)	100 - (10 / 30) × 100 = 66 %	

표 6 완성형 트루타입 폰트의 2 단계 압축 결과

	12 × 12	14 × 14	16 × 16	24 × 24	결 과
압축 전 (byte)	13,824	19,200	24,576	55,296	112,896
압축 후 (byte)	13,824	15,945	18,489	24,725	72,983
압축률 (%)	0 %	17 %	25 %	55 %	35 %

표 7 조합형 트루타입 폰트의 2 단계 압축 결과

	11 × 11	12 × 12 / 13 × 13	15 × 15	결 과
압축 전 (byte)	×	247,026	256,750	503,776
압축 후 (byte)	×	233,115	235,941	469,056
압축률 (%)	×	6 %	8 %	7 %

되는 오버헤드가 증가하기 때문이다. 이를 해결하기 위해서는 비트맵을 나눌 때 자신을 가리키고 있는 합성 비트맵의 수를 확인하여 일정 수 이상일 때는 비트맵을 나누지 않는 방법과, 나누어진 결과를 이전의 결과와 비교하여 비트맵을 작은 단위로 나눈 것이 더 많은 크기를 차지할 경우, 다시 복구하는 기능을 추가하여야 한다. 또한 한자에서는 본 논문의 알고리즘으로 예상했던 부수 정보를 추출하기가 어렵다. 왜냐하면 한글은 초성, 중성, 종성으로 명확히 구분되지만, 한자의 경우는 부수의 위치를 예측하기가 어렵고 비트맵을 구성하는 점의 밀집도에서도 한글보다 높기 때문에 8방향 탐색을 이용해서 각각의 부수를 추출해 내기가 힘들다. 그렇기 때문에 한자의 부수 정보를 추출하기 위해서는 각각의 글자마다 부수의 위치 정보를 미리 알고 접근해야 한다.

본 논문에서는 압축을 위한 두 단계의 압축과정을 제시하였다. 하지만 트루타입 폰트의 형태와 방식에 따라 다른 압축률을 보이고 있다. 이는 각각의 형태에 대해 최적화된 알고리즘의 필요성을 요구하며, 향후 추가적인 압축방식에 대한 연구를 통해 보다 많은 압축결과를 기대할 수 있다. 또한 한자의 경우는 조합형 형태의 트루타입 폰트가 없기 때문에 부수 정보 등을 추출하여 합성 비트맵을 구성한다면 많은 압축률을 보일 것이다[15].

참 고 문 헌

[1] 김은희, 정근호, 최재영, "트루타입의 합성 글립을 이용한 한글 폰트의 중복성 최소화 방법", 정보과학회논문지(B), 제26권, 제10호, 1999.

[2] Richard Rubinstein, "Digital Typography, An Introduction to Type and Composition for Computer System Design," Addison Wesley, 1988.

[3] Roger D. Hersch, Ecole Polytechnique Federale, and Lausanne, *Visual and Technical Aspect of Type*, pp.110-125, Cambridge University Press, 1993.

[4] Adobe System, "Font Feature File Processing," United States Patent 6,426,751, 2002.

[5] Microsoft Corp., TrueType 1.0 Font Files, Microsoft Corp., Nov. 1995.

[6] Peter Karow, "Font technology, Methods and Tools," ISBN/EAN: 0387572236, Springer-Verlag, 1994.

[7] 이영표, "합성글립을 이용한 트루타입폰트 손실 최적화기 구현", 숭실대학교 컴퓨터학과 석사학위논문, 2000.

[8] Peter Karow, "Digital Typefaces: Description and Formats," ISBN: 0387565094, Springer-Verlag, 1994.

[9] Microsoft Windows.CE, available, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wcepb40/html/pboriUsingPlatformBuilder.asp.

[10] Agfa, "Method for Data Compression of Digital Data to Produce a Scaleable Font Database." United States Patent 5,754,187, 1994.

[11] 임순범, "글꼴 처리 기술의 발전 동향", ISBN: 8982755802, (주) 세종대왕기념사업회, 1999.

[12] 안은영, 조형제, "기본 획 합성에 의한 한글글꼴 생성", 정보과학회 논문지, 제21권, 제4호., pp.649-651, 1994.

[13] 이준희, 정내권, "컴퓨터속의 한글," (주) 정보시대, pp.457-473, 1991.

[14] 오길록, 최기선, 박세영, "한글공학", 대영사, 1995.

[15] Microsoft Typography, available, http://www.microsoft.com/typography.

한 주 현

2000년 숭실대학교 컴퓨터학부(공학사) 2002년 숭실대학교 대학원 컴퓨터학과(석사). 2002년~현재 숭실대학교 대학원 컴퓨터학과 박사과정. 관심분야는 유비쿼터스 컴퓨팅, 분산/병렬처리, 시스템 소프트웨어, 시스템 보안, 한글폰트기술

정 근 호

1993년 숭실대학교 전자계산학과(학사) 1995년 숭실대학교 전자계산학과(석사) 1995년~2000년 숭실대학교 컴퓨터학과 박사과정 수료. 1997년~2000년 (주)상지소프트 팀장. 2000년~현재 (주)네오퍼스 CTO. 관심분야는 폰트처리, 한글공학, 전자문서

최 재 영

1984년 서울대학교 제어계측공학과(공학사). 1986년 미국 남가주대학교 컴퓨터공학과(석사). 1991년 미국 코넬대학교 컴퓨터공학과(박사). 1992년 1월~1994년 2월 미국 국립 오크리지연구소 연구원 1994년 3월~1995년 2월 미국 테네시 주립대학교 연구교수. 1995년 3월~현재 숭실대학교 컴퓨터학부 부교수. 관심분야는 유비쿼터스 컴퓨팅, 그리드 컴퓨팅, 병렬/분산처리, 클러스터링, 시스템 소프트웨어