임베디드 시스템에서의 폰트 처리 기술

(주)네오퍼스 정근호 · 허 길 · 이영표
숭실대학교 최재영*

1. 서 론

최근 들어 PDA, 셋톱박스, 전자책 등 비주얼이 강조되는 임베디드 시스템들이 많이 개발되고 있고, 그러한 시스템의 사용자들도 점차적으로 늘어나고 있다. 본 고에서는 임베디드 시스템에서 폰트를 - 특히 한글과 한자를 포함하는 동양권 폰트를 - 처리하는 과정에서의 문제점들과 최신 기술 동향을 알아본다.

일반 PC와 비교하여 임베디드 시스템은 프로세서 속도가 느리고 저장 공간이 제한되어 있으므로, 일반 PC에서 사용하는 폰트 처리 기술을 그대로 적용하기가 용이하지 않다. 그리고 영숫자와 기호의 경우 200자 내외만을 처리하면 되지만, 한글과 한자를 사용하는 동양권에서는 처리해야 할 글자의 수가 2만~3만자 정도까지 늘기 때문에 영어만을 처리하는 것보다 빠른 처리 속도와 많은 저장 공간을 요구한다. 따라서 임베디드 시스템에서 동양권 글꼴을 효과적으로 처리하기 위해서는 추가적으로 고려해야 할 사항들이 있다.

먼저 글꼴의 분류와 특성을 알아보자. 컴퓨터로 표현되는 글꼴 방식에는 기본적인 표현 단위의 대상에 따라 크게 점 글꼴(bitmap font), 윤곽선 글꼴(outline font), 그리고 구조적 글꼴(structured font)의 세 가지로 분류할 수 있다 [1]. 그림 1은 각 글꼴의 예를 보여주고 있다.

(a) 점글꼴 (b) 윤곽선 글꼴 (c) 구조적 글꼴

그림 1 점 글꼴과 윤곽선 글꼴, 구조적 글꼴의 예

프린터나 스크린 등의 출력 장치에 최종적으로는 점들을 찍음으로써 글자를 출력하는데, 이 점들을 그대로 표현한 방식이 점 글꼴이다. 컴퓨터에서 가장 간단하게 글자를 표현하는 방식으로, 글자의 내부에 해당하는 부분을 1로, 외부를 0으로 하여, 한 글자에 해당하는 사각형을 점 행렬(dot matrix)로 기억한다. 이 방식은 설계할 때의 모양과 출력된 글자의 모양이 같으므로 설계자의 의도대로 쉽게 설계할 수 있다. 또한 제작 시간이 적게 걸리고, 적은 저장 공간을 필요로 한다. 해상도가 낮은 출력 장치에서는 매우 효과적이지만, 글자를 확대할 경우 글자의 윤곽이 거칠어지고, 기울임이나 임의의 각도로 글자를 출력하기 어려운 단점이 있다.

윤곽선 글꼴은 글자의 윤곽을 여러 부분으로 나누어 직선, 원호, 자유 곡선 등으로 표현한다. 자유 곡선으로는 3차 운형(cubic spline) 곡선, 베지어(Bezier) 곡선, 2차 B-운형(quadratic B-spline) 곡선 등이 주로 사용된다. 가장 널리 사용되는 마이크로소프트의 트루타입(TrueType) 글꼴의 경우 2차 B-운형 곡선을 사용하며, 어도비(Adobe)의 Type 1 글꼴은 베지어 곡선을 사용하고 있다. 윤곽선 글꼴은 하나의 크기에 대한 글자 데이터를 가지고 자유롭게 임의의 글자 크기나 기울임, 각도 등에 대해 처리가 가능하므로 기억 용량이 매우 감소하지만, 획의 굵기 변화 등 글꼴 모양 자체의 변화까지 처리할 수는 없다.

또한 윤곽선 글꼴은 출력 글자를 생성할 때는, 일단 출력하는 크기에 맞추어 윤곽선의 직선이나 곡선 부분을 좌표로 계산하고, 그 내부를 채우는 과정이 필요하므로 속도가 느려진다. 이러한 계산 과정 때문에 70년대까지는 실용화되지 못하다가, 80년대 중반 이후 고속 프로세서가 개발되면서 고가의 레이저 프린터 등에서 사용되기 시작하였다. 90년대에는 컴퓨

* 종신회원

터의 속도 향상이 더욱 가속되어, 오늘날에는 거의 모든 출력기에서 대부분 윤곽선 글꼴의 글자를 실시간으로 출력하여 보여준다.

구조적 글꼴은 주로 글자 획의 중심선(stroke skeleton)을 이용하여 글자의 모양을 표현한다. 대표적으로 D. E. Knuth가 개발한 Metafont 시스템이 있다. 개별적인 글자는 물론이고 한 벌 전체의 글꼴 특성을 고려하여 글꼴을 쉽게 설계할 수 있다. 또한 하나의 글꼴 데이터에서부터 획의 굵기 변화 등 다양한 변형에 대해 자동 처리가 가능하다. 구조적 글꼴을 사용하면 데이터의 양이 크게 줄어들고 글꼴의 설계나 글꼴의 변형이 아주 용이하지만, 글꼴을 생성하는 속도가 매우 느리다 [2]. 상대적으로 느린 처리 속도로 인하여 글자의 출력 과정에서는 사용되지는 못하고, 컴퓨터에서 글자를 설계하는 곳에서만 일부 응용될 뿐이다.

이상에서 알아본 것처럼, 점 글꼴은 여러 단점을 가지고 있고 구조적 글꼴은 처리 속도가 매우 느리기 때문에, 오늘날 대부분의 컴퓨터 시스템에서는 일반적으로 윤곽선 글꼴을 사용하고 있다. 또한 임베디드 시스템에 내장된 프로세서의 성능과 출력 장치의 해상도가 윤곽선 글꼴을 사용할 수 있을 정도로 발전하였고, 실제 윤곽선 글꼴을 사용하는 경우도 늘어나고 있다. 본 논문에서는 윤곽선 글꼴에 초점을 두고 이를 효과적으로 처리하여 출력시키기 위한 래스터라이저, 그리고 해상도가 낮은 화면에서 폰트의 품질을 향상시키기 위한 회색조 폰트 처리 방법과 부분 픽셀 렌더링 기술을 살펴본다. 마지막으로 동양권 폰트의 저장 공간을 절약하기 위한 폰트 압축 및 재구성 기법을 기술한다.

2. 래스터라이저

윤곽선 글꼴 데이터는 특정 크기의 마스터 글자에 대한 곡선 제어점의 좌표 값만 가지고 있다. 이들 좌표 값으로부터 최종 출력되는 글자의 비트맵 이미지를 만들어내야 하는데, 이 과정을 처리하는 프로그램을 래스터라이저(rasterizer)라고 한다. 래스터라이저는 출력 품질과 직결되므로, 대부분의 서체나 전자출판과 관련된 업체에서는 저마다 독특한 고품질의 래스터라이저를 확보하고 있다. 포스트스크립트(Postscript) 서체의 원천적인 기술을 보유하고 있는 어도비(Adobe)의 타입매니저(ATM)를 비롯하여, 애플(Apple)의 퀵드로우(QuickDraw), 마이크로소프트의 32 bit Rasterizer 등이 대표적인 래스터라이저들이다.

내장형 시스템에서 폰트를 출력시키려면 서체 자체와 함께 래스터라이저도 라이센스하여야 하며, 이는 내장형 시스템 제품의 생산 비용과 직결된다. 서체와 래스터라이저에 대한 원천 기술을 확보하고 있는 회사들은 일반적으로 서체와 래스터라이저를 하나의 패키지로 판매하며, 제품당 일정한 라이센스 비용을 요구한다. 예를 들어 포스트스크립트 서체의 기술을 보유하고 있는 어도비는, 레이저 프린터를 생산하는 모든 회사들로부터 프린터마다 일정 금액의 비용을 받는 조건으로 자사의 포스트스크립트 서체와 이를 위한 래스터라이저를 라이센스하고 있다.

내장형 시스템에서는 시스템용 운영체제와 래스터라이저를 통합함으로써, 운영체제 단위로 판매하는 경우도 있다. 이 경우에는 운영체제만을 구입하여 제품에 탑재함으로써 래스터라이저에 대한 추가적인 비용 지출을 막을 수 있지만, 래스터라이저만을 다른 것으로 대체하기가 용이하지 않기 때문에 다양한 서체 타입의 지원이 어려워진다. Windows CE로 내장형 시스템 시장에 뛰어든 마이크로소프트는 내장형 시스템용 운영체제와 래스터라이저를 통합한 대표적인 예이다.

래스터라이저는 그림 2에서 보듯이 폰트를 가져오는 부분 (폰트 로더: Font Loader), 가져온 폰트 형식으로부터 글자 모양을 만들어내는 부분 (글꼴 모양 생성기: Glyph Image Generator), 그리고 만들어낸 글자를 래스터 장비에 출력하기 위하여 적용시키는 부분 (렌더링 처리 모듈 : Rendering Processing Module)으로 구성된다 [3].

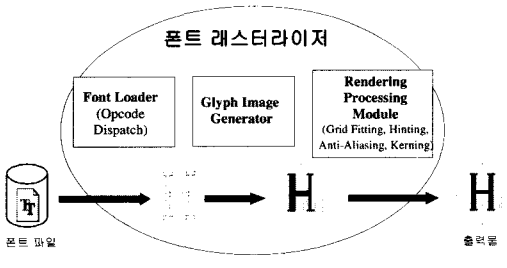


그림 2 래스터라이저의 일반적인 구조

폰트 로더는 폰트 이미지로부터 필요로 하는 글자

에 대한 정보를 가져오는 부분이다. 폰트 이미지는 디스크나 파일로 저장되어 있거나 ROM에 내장되어 있기도 하며, 혹은 실시간으로 네트워크를 통해서 다운로드 받을 수도 있다. 이 과정은 래스터라이저의 전체 처리 시간의 대부분을 차지하는데, 하드웨어 특성상 폰트 파일이나 ROM으로부터의 입출력 과정에서 걸리는 시간이 다른 부분보다 훨씬 많이 소요되기 때문이다.

영어권에서는 폰트 캐쉬 등의 방법을 이용하여 이 과정을 개선하려는 연구가 많이 진행되었지만, 많이 적용되고 있지는 않다. 그 이유는 영문 폰트의 경우, 폰트 이미지의 크기가 워낙 작기 때문에 고도의 기술을 사용하지 않더라도 성능이 좋기 때문이다. 하지만 2 Bytes 문자 체제를 사용하는 동양권에서는 폰트의 크기가 훨씬 크기 때문에, 영문 폰트를 사용할 때와 같은 좋은 성능을 기대할 수 없다. 따라서 동양권 2 Bytes 문자 체제에 적합한 효과적인 폰트 캐쉬 기술이 요구된다.

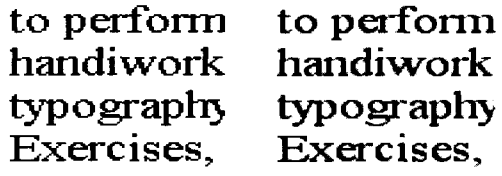
글꼴 모양 생성기는 폰트 이미지로부터 가져온 글자에 대한 정보와 출력 환경에 맞도록 필요로 하는 여러 가지의 글자 속성 정보를 바탕으로 글자의 모양을 계산하여 그려낸다. 일반적인 폰트는 글꼴 모양에 명령어 코드(opcode)의 집합으로 구성되어 있기 때문에, 얼마나 빨리 명령어를 해독하여 신속하게 그려낼 수 있는가가 중요하다.

글자 모양 생성기에서 그려진 아날로그적인 글자 모양을 가로, 세로가 고정된 저해상도 장비에서 정확하게 나타내기는 어렵다. 렌더링 처리 모듈에서는 실제 화면에 출력시키기 위해 글꼴 모양 생성기에서 만들어진 수학적인 곡선을 적절하게 변환시킨다. 안티에일리어싱(Anti-Aliasing), 힌팅(Hinting), 커닝(Kerning) 등과 같은 처리 과정을 거쳐서 저해상도 화면에서 처음 글자 모양과 가깝게 출력하도록 보정한다 [4]. 또한 성능을 높이기 위해서는 최종으로 만들어진 글립(glyph) 이미지를 적절하게 캐슁하여, 불필요하게 중복되는 폰트 입출력이나 폰트 모양을 만들어내는 과정을 줄여서 성능을 극대화하도록 한다. 글자의 출력 품질을 좌우하며, 실제 래스터라이저의 상품성을 결정짓는 가장 중요한 부분이다. 사실상 가장 많은 노하우와 기술을 필요로 하며, 관련 회사마다 자체 노하우를 확보하기 위하여 연구하고 있다.

3. 화면용 글꼴 처리 기술

PDA, 셋톱박스, 전자책 등의 임베디드 시스템에서의 출력 장치는 주로 CRT나 LCD 등의 화면이다. 보통 300~1,200 dpi의 높은 해상도를 갖는 프린터와 달리, 이러한 출력장치는 75~100 dpi가 일반적이고 최고 200 dpi 정도의 해상도를 갖기 때문에, 종이에 인쇄된 출력물에 비해 가독성이 많이 떨어진다. 요즈음 많이 사용되는 단말기의 화면은 흑백 이외에 최소 256개 또는 65,000개 이상의 색상을 나타낼 수 있다. 또한 칼라 색상 이외에 회색조(gray scale)를 나타낼 수 있는 범위도 몇십 가지에 이른다.

이러한 화면의 특성을 이용하여 글자의 테두리에서 계단 현상이 일어나는 부분에 회색 처리를 하면 글자의 테두리가 부드럽게 보인다. 또한 매우 작은 크기 글자의 경우 흑백으로는 글자의 판독이 어렵지만, 회색조 처리를 하면 어느 정도 글자의 판독이 가능해진다. 이러한 기술이 적용된 글꼴을 회색조 글꼴(gray scale font), 부드러운 글꼴(smooth font), 또는 계단 방지 글꼴(anti-aliased font)이라고 부른다.



(a) 점 글꼴 (b) 회색조 글꼴

그림 3 회색조 글꼴의 예

회색조 글꼴은 래스터라이저의 주사선을 변환하는 과정에서 만들어진다. 주사선을 변환할 때 비트맵상의 한 픽셀을 흑백에 해당하는 1 또는 0의 한 비트 값으로 구하는 것이 아니라, 회색조의 단계 값으로 구하게 된다. 가령 16 단계의 회색조 글꼴이라고 하면 각 픽셀은 0부터 16사이의 값을 가지며, 완전히 검은 색이 16이고 중간의 회색 값은 그 농도에 따라 1부터 15까지의 값을 갖는다. 따라서 주사선 변환 과정에서 구하는 것은 각 픽셀이 포함되는지의 여부를 구하는 것이 아니라, 그 픽셀을 중심으로 포함되는 비율을 구해야 한다.

1998년 11월 미국의 컴덱스 전시회에서 마이크로소프트는 전자도서용 LCD 화면에서 사용할 수 있는 글꼴 기술인 ClearType과, 이 글꼴 기술을 적용한 Microsoft Reader를 발표하였다. ClearType 기술은 화면용 글꼴 처리에 있어서 일반적인 회색조 글꼴보

다 LCD 화면의 특성을 살린 글꼴 처리 기술을 적용한 것이다 [5].

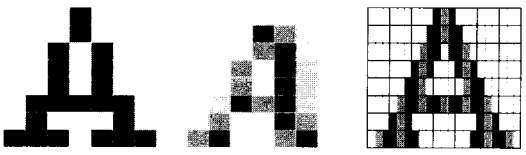
보통 사람의 눈으로 볼 때 LCD 화면의 한 픽셀은 픽셀 하나로 구성되어 있는 것처럼 보이지만, 실제로는 RGB 세 가지 색상의 픽셀이 모여있다. 이들 간격이 매우 작기 때문에 사람 눈에는 착시 현상으로 한 픽셀처럼 보이는 것이다. 이 때 각 색상을 가지는 픽셀의 크기는 1/3이 되므로, 그림 4 (b)에서처럼 한 픽셀은 세 개의 가로선, 즉 RGB의 부분 픽셀로 구성되어 있다고 볼 수 있다. 이러한 부분 픽셀의 크기를 이용하여 가로 방향의 해상도를 높이는 기법을 적용한 것이다.

(a) 눈으로 보이는 한 픽셀/줄

(b) 실제 출력되는 부분 픽셀들

그림 4 LCD 화면의 부분 픽셀

만약 해상도 자체가 가로 방향으로 세 배가 된다면, 래스터라이저가 주사선을 변환할 때 가로 폭이 세배인 비트맵을 만들어주어 경사 부분인 훨씬 부드러운 글자의 출력이 가능하다. 실제 해상도가 세 배가 아닌 부분 픽셀인 경우에도 가로 폭이 세 배인 비트맵을 만들어 준 후, 각 부분 픽셀의 칼라 명암에 따라 비중을 곱하여 값을 결정해 주면, 마찬가지로 부드러운 글자 출력 결과를 얻을 수 있다. 이 때 출력되는 글자의 테두리는 검은색 또는 흰색이 아니라 임의의 칼라 색상을 가지게 되는데, 확대하여 보기 전에는 착시 현상으로 인하여 흑백 또는 회색으로 보이게 된다. 이러한 부분 픽셀 글꼴의 출력 결과는 그림 5에서 보듯이 회색조 글꼴보다 훨씬 더 나은 품질을 보이고 있다[6].



(a) 점 글꼴 (b) 회색조 글꼴 (c) 부분 픽셀 글꼴

그림 5 회색조 글꼴과 부분 픽셀 글꼴

글자 모양의 특성상 가로 해상도가 세로 해상도보다 출력 결과에 더 큰 영향을 미친다. 또한 가로 방향의 해상도 증가는 이탤릭체나 경사체에서 그 장점이 더욱 돋보이며, 작은 크기의 글자간 간격(kerning)이나 굵은 글자 효과 등을 보다 세밀하게 조정할 수 있다. 그림 6의 출력 결과와 같이 개별 글자 하나뿐 아니라 전체 문장의 가독성도 매우 좋아지게 된다.

Here we stand, on the brink of a significant step forward in ***Personal Computing Evolution:*** *Old and mature display technology is merging with modern LCD display panels to deliver breathtaking new clarity and elegance to the written word.* ***Indeed . . . this changes the world!*** *(right?!!)*

Here we stand, on the brink of a significant step forward in ***Personal Computing Evolution:*** *Old and mature display technology is merging with modern LCD display panels to deliver breathtaking new clarity and elegance to the written word.* ***Indeed . . . this changes the world!*** *(right?!!)*

그림 6 부분 픽셀 글꼴이 적용된 문장의 출력 결과

부분 픽셀은 가로 방향으로 배치되어 있으므로, 세로 방향으로는 해상도 증진 효과가 없다. 따라서 경사가 낮은 사선부분에서는 거의 효과가 나타나지 않는다. 부분 픽셀 글꼴은 글자의 설계와 관계가 있는 것이 아니라 출력 장치의 특성을 최대한 이용한 주사선 변환 기술이므로, LCD 화면에서만 효과를 얻을 수 있으며, CRT 화면에서는 그러한 효과를 기대할 수 없다.

4. 폰트 압축 기술

영문 폰트 파일의 크기는 수 Kbyte로, 한글이나 한자에 비해 매우 작기 때문에 문제가 되지 않지만, 동양권 폰트 파일의 크기는 수 Mbyte 이상으로 많은 저장 공간과 빠른 처리 속도가 요구된다. 폰트 크기를 압축하여 저장 공간을 최소화하기 위한 대표적인 기술로 아그파(Agfa)의 비손실 압축 기술과 네오퍼스의 트루타입 폰트 손실 압축 기술을 알아본다.

먼저 아그파의 비손실 압축 기술은 폰트 화일의 데이터들을 테이블별로 분석하여, 각 테이블별로 자료값의 분포에 따라 폰트를 미리 압축해 놓았다가, 필요할 때 압축된 파일을 복원하는 방식을 사용한다.

그림 7은 압축 및 복구 과정과 해당 데이터 파일들

을 보여주고 있다. D1은 트루타입 폰트 파일이고, D2는 이를 대신하는 압축 데이터 폰트 파일이며, C1은 이를 생성하기 위한 압축 메커니즘이다. 복원 메커니즘인 C2는 압축된 데이터 파일인 D2를 입력으로 받아 트루타입 폰트 파일인 D3를 생성한다. 압축 과정에서 불필요한 정보를 제거하기 때문에 D1과 D3는 동일한 파일이 아닐 수도 있지만, 트루타입 폰트로써 기능적으로는 동일하다.

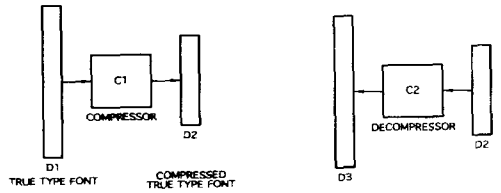


그림 7 폰트의 비손실 압축과 복원

아그파의 비손실 폰트 압축 기술은 폰트 테이블별로 별도의 압축 알고리즘 적용, 재구성, 전체 자료 압축의 단계를 거친다. 먼저 폰트 파일을 구성하는 각 부분들을, 각 부분에 적합한 특수한 압축 방법을 이용하여, 중간 압축 형식으로 만든다. 다음으로 각각 다른 방식으로 압축된 폰트 파일 부분들을 분리해서 중간 압축 형식을 재구성한다. 마지막으로 재구성된 중간 압축 형식에 다시 일반 압축 알고리즘을 적용해서, 최종적으로 압축된 폰트 파일인 CTF (Compressed TrueType Font)를 생성한다 [7].

비손실 압축 기술을 적용하면 한글과 한자를 포함하는 동양권 글꼴의 경우 평균적으로 원래 크기의 70%로 줄일 수 있다. 그러나 글꼴을 사용할 때 별도의 복원 과정이 필요하기 때문에 별도의 프로세서 처리 시간이 필요하다. 또한 압축된 파일을 저장하는 저장 공간은 절약되지만, 압축 화일을 복원한 후에 글꼴을 사용할 수 있기 때문에 글꼴 처리에 있어서의 작업 공간은 압축 이전과 유사하다.

트루타입에서의 글립은 단순 글립(Simple Glyph)과 합성 글립(Composite Glyph)의 두 가지가 있다. 단순 글립은 실제로 화면에 나타날 폰트들의 좌표 값들로 구성되어 있으며, 윤곽선 좌표를 나타내는 점들과 폐곡선을 나타내는 contour들로 구성되어 있다. 합성 글립은 다른 글립들을 참조하기 위하여, 참조하려는 글립의 인덱스와 그 글립의 위치를 지정하고 크기를 변경할 수 있는 테이블들로 구성되어 있다. 트루타입 폰트 파일에는 저작권, 폰트 이름, 사용 허가권, 문자에 대한 기술 정보, 힌팅 명령어, 문자 집합과 매핑 등의 많은 정보가 테이블 형식으로 저장되어 있다. 폰트 파일은 최대 24개의 테이블로 구성되어 있으며, 이중 10개는 반드시 필요하고 나머지 14개 테이블은 선택적이다 [8]. 이 테이블 중에서 glyf 테이블은 실제로 문자 정보가 들어가고, 각 글립에 대한 윤곽선 갯수, x, y축에서의 좌표 값, 힌팅 프로그램, 글립을 구성하는 좌표점들, 그리고 각 좌표점들에 대한 플래그들로 구성되어 있다. 따라서 glyf 테이블은 모든 글립에 대한 실제 정보를 유지하고 있으므로, 이 테이블의 크기에 의해서 트루타입 폰트 저장 공간의 크기가 좌우된다.

네오퍼스의 손실 압축 기술은 일정 부분에서의 오차를 무시하고 유사 글립들을 통합하여 재구성하는 방식을 사용하여 트루타입 폰트의 크기를 줄인다. 압축 과정은 이론적으로는 아주 간단하다. 먼저 트루타입의 전체 테이블을 읽으면서, 글립 테이블을 contour 단위로 나눈다. 분리한 contour끼리 비교하면서 유사한 contour들을 찾는다. 유사성을 비교할 때에 각 폰트의 출력 해상도를 고려하여 contour 좌표간의 오차 허용 범위를 결정한다. 오차 허용 범위에서 유사하게 나타난 contour 끼리는 하나의 리스트로 병합하고, 대표 contour를 선택한다. 유사성 리스트에 있는 다른 contour들이 대표 contour를 참조하도록 글립 테이블을 재구성하고, 마지막으로 전체 테이블을 병합한다 [9].

그림 8은 contour간의 유사성을 비교하는 과정을 보여준다. "가"와 "갸"를 XOR로 연산한 후에, 두 글자에서의 "ㅏ"에 대한 유사성을 나타내고 있다. 그림 8 (c)에서 원은 각 contour의 좌표에 대한 오차의 허용 범위를 나타내고, 원 내에 점이 포함되어 있으면, 오차의 허용 범위 내에 있다는 것을 의미한다.

(a) '가' (b) '갸' (c) '가' XOR '갸'

그림 8 나눈 contour간의 유사성 비교

표 1은 마이크로소프트 WinCE 플랫폼 빌더에서

제공하는 4 종류의 한글 폰트를 24 dot에서 동일한 결과를 출력하도록 오차 범위를 설정하고 손실 압축했을 때의 압축률을 보여준다. 굴림과 바탕에 있는 유니코드 한글 (11,172자 포함)의 경우에서는 74%의 압축률을, 돋음에 있는 완성형 한글 (2,350자 포함)의 경우 40%의 압축률을 보이고 있다. 헤드라인의 경우 유니코드를 포함하고 있지만, 폰트 구조가 단순하고 폰트의 크기 자체가 처음부터 작기 때문에 47%의 압축률을 보인다. 한자의 경우 4,880자를 사용하고 있으며, 한자의 구성이 한글보다 다양한 크기와 위치를 갖기 때문에 압축률이 다소 떨어지고 있다. 만일 한자의 각 획을 미리 분리해내어 비교한다면 압축률을 더 높일 수 있을 것이다. 이러한 손실 압축 기술은 폰트의 재구성 기법을 사용하기 때문에, 별도의 복원 과정이 필요없으며, 래스터라이저를 변경하거나 시스템을 재설정하지 않고, 압축하지 않은 원래의 폰트를 사용하는 방법과 동일한 방법으로 폰트를 처리할 수 있다.

표 1 압축률 비교 테이블
(24 dot 글씨 기준, 단위 KB)

폰트 이름	범위	원래 크기	압축된 크기	비율 (%)
굴림	한글	5,838	1,543	26.4
	한자	4,597	4,065	88.4
	전체	10,142	5,608	55.3
바탕	한글	10,751	2,844	26.5
	한자	3,459	2,544	73.7
	전체	13,883	5,361	38.6
돋움	한글(완성형)	1,268	756	59.6
	한자	2,310	1,782	77.1
	전체	3,298	2,258	68.5
헤드라인	한글	1,170	625	53.4
전체		28,467	13,824	48.7

5. 맺음말

비주얼이 강조되는 임베디드 시스템에서의 글꼴 처리와 관련된 기본적인 개념들과 래스터라이저, 글꼴 품질 향상을 위해 사용되는 회색조 글꼴과 부분 픽셀 렌더링 기술, 저장 공간을 절약하기 위한 폰트 압축 기술들을 살펴보았다.

래스터라이저는 폰트 로더, 글꼴 모양 생성기, 렌더링 처리 모듈로 구성되며, 그 성능은 시스템의 출력 품질과 속도를 크게 좌우한다. 특히 2바이트 문자 체계를 사용하는 경우, 처리할 글꼴의 종류와 규모에 따른 적절한 글꼴 캐쉬의 설정이 필수적이다. 저해상도의 출력장치를 위해서는 안티 에일리어싱, 힌팅, 커닝 등의 보정을 통해 출력 품질을 높이게 된다. 공간과 프로세서의 성능이 제약된 대부분의 임베디드 시스템에서 반드시 고려해야 할 요소들이다.

화면용 글꼴의 품질 향상을 위해서 회색조 글꼴과 부분 픽셀 렌더링 기술이 사용된다. 부분픽셀 렌더링은 상당한 처리 시간을 요구하지만, 점차로 높은 성능의 프로세서를 사용하므로 폰트의 품질을 크게 향상시킬 수 있다. 하지만 LCD 화면의 특성을 이용하기 때문에 LCD 이외의 출력 장치에서는 회색조 글꼴만을 사용할 수 있다.

임베디드 시스템에서의 저장 공간과 처리 공간의 제약과 매우 큰 동양권 글꼴, 다수의 글꼴에 대한 요구 사항들로 인한 글꼴의 압축은 필수적이다. 글꼴 압축 기술은 비손실 압축과 손실 압축으로 구분된다. 비손실 압축은 자료의 통계적 분포에 기반하여 폰트의 각 부분을 인코딩한 후에 각각에 맞는 압축 알고리즘을 적용하였으며, 손실 압축은 글꼴의 유사성에 기반하여 폰트 파일을 재구성하는 기법을 사용하였다. 비손실 압축은 항상 동일한 출력 품질을 보장하지만, 낮은 압축율을 가지며 또한 부가적인 복원 과정을 거쳐야 한다. 하지만 손실 압축은 오차 범위 이내에서 동일한 출력을 보장하고 보다 높은 압축율을 보여준다, 복원 과정에서의 오버헤드를 감수할 수 있는 경우에는 비손실 압축과 손실 압축 기법을 동시에 적용하여 전체 글꼴의 크기를 원래 크기의 35% 수준으로 줄일 수도 있다.

폰트를 처리하는 방법은 사용될 시스템의 하드웨어뿐 아니라, 사용하는 문자의 영향을 많이 받는다. 특히 프로세서의 속도가 느리고 사용하는 공간이 제약된 임베디드 시스템에서는 사용하는 문자가 전체 시스템의 폰트 출력 품질과 처리 속도를 좌우하게 된다. 대부분의 폰트 처리 기술들은 주로 영문 폰트들을 처리하기 위해 개발되었으며, 한글과 한자를 포함하는 동양권 폰트 처리에도 거의 그대로 사용하고 있다. 프로세서 속도가 느리고 저장 공간이 제한되어 있는 임베디드 시스템에서 빠른 처리 속도와 많은 저장 공간을 필요로 하는 동양권 폰트들을 효과적으로 처리하기 위한 새로운 기술들이 요구되고 있다.

참고문헌

[1] 임순범, "윤곽선 글자꼴의 처리 기술 및 활용 추세," 전자 공학회지, 제18권, 제11호, 1991년 11월, pp.858-865.
[2] Richard Rubinstein, Digital Typhography, Addison-Wesley, pp.140-147, 1998.
[3] David Turner, "The design of FreeType 2," http://www.freetype.org/freetype2/docs/design/index.html, 2000.
[4] TrueType Typography, "TrueType Hinting," http://www.truetype.demon.co.uk/tthints.htm, 2001.
[5] Microsoft Corporation, "About Microsoft Typography," http://www.microsoft.com/typography/default.asp, 1999.
[6] Gibson Research Corporation, "Sub-Pixel Font Rendering Technology," http://www.grc.com/cleartype.html, 1999.
[7] Agfa Corporation, "Method and Apparatus for Font Compression and Decompression," United States Patent 6,031,622," Feb. 2000.
[8] Microsoft Corporation, "TrueType 1.0 Font Files Technical Specification, Revision 1.66," Nov. 1995.
[9] 김은회, 정근호, 최재영, "트루타입의 합성 글립을 이용한 한글 폰트의 중복성 최소화 방법," 정보화학회 논문지(B), 제26권, 제10호, pp.1230-1236, 1999.

정 근 호

1993 숭실대학교 전자계산학과 학사
1995 숭실대학교 전자계산학과 석사
1995~현재 숭실대학교 컴퓨터학과 박사과정 수료
1997~2000 (주)상지소프트 팀장
2000~현재 (주)네오퍼스 CTO
관심분야:폰트처리, 한글공학, 프로그래밍 언어
E-mail:gnho@neopus.com

허 길

1995 단국대학교 전자계산학과 학사
1997 숭실대학교 컴퓨터학과 석사
1997~1999 (주)세기통신 개발부 팀장
1999~현재 숭실대학교 컴퓨터학과 박사과정 수료
2000. 8~현재 (주)네오퍼스 팀장
관심분야:내장형 시스템, 실시간 시스템, 폰트처리
E-mail:semylab@neopus.com

이 영 표

1998 숭실대학교 컴퓨터학과 학사
2001 숭실대학교 컴퓨터학과 석사
2001~현재 (주)네오퍼스 팀장
관심분야:폰트처리, Mobile computing.
E-mail:neofrog@neopus.com

최 재 영

1984 서울대학교 제어계측공학과 학사
1986 미국 남가주대학교 컴퓨터공학 석사
1991 미국 코넬대학교 컴퓨터공학 박사
1992~1994 미국 오크리지연구소 연구원
1994~1995 미국 테네시주립대학교 연구교수
1995~현재 숭실대학교 컴퓨터학부 조교수/부교수
관심분야:시스템 소프트웨어, 폰트처리, 병렬/분산처리
E-mail:choi@comp.ssu.ac.kr