工學碩士 學位論文

휴대전화 단말기를 위한 개선된 이미지 처리 알고리즘의 구현

Implementation of Advanced Image Processing Algorithm for Cellular Phone

指導敎授 林 宰 弘

2003年 2月

韓國海洋大學校 大學院

電子通信工學科

崔 宰 榮

工學碩士 學位論文

휴대전화 단말기를 위한 개선된 이미지 처리 알고리즘의 구현

Implementation of Advanced Image Processing Algorithm for Cellular Phone

指導敎授 林 宰 弘

2003年 2月

韓國海洋大學校 大學院

電子通信工學科

崔 宰 榮

本 論文을 崔宰榮의 工學碩士
學位論文으로 認准함.

委員長　梁　圭　植 (印)

委　員　朴　東　國 (印)

委　員　林　宰　弘 (印)

2003年　2月

韓國海洋大學校　大學院

電子通信工學科　崔　宰　榮

목 차

그림 목차

표 목차

Abstract

In this thesis, improvement algorithms of raising the memory efficiency through encoding or decoding the large size image of cellular phones are designed and realized.

Most of cellular phone uses bit mapped image in the memory with the value of hexadecimal, which is proper for rapid change and display of data.

However, It is needed the high quality of graphics in the cellular phone especially in the multimedia era. The improvement in the hardware and software technology made it possible to develope the cellular phone with 256 or 65,000 colors.

On the side of memory as an example, 6.4 Kbytes of memory are needed to store the image of 101 * 80 size with 256 colors.

Data compress technique is of interest, because cellular phone which uses a kind of operation system have limited memory and unit prices of the memory chips have significant effect on marketing.

Therefore, in this thesis, algorithms are proposed to decode LZW data and display image on LCD like a bitmap image through using the LZW compress algorithm which is used in GIF file. As a result of LZW compression of the bitmap image, proposed LZW(Lempel-Ziv-Welch) algorithm show the reduction of memory size as benefit of image size when convert bitmap image to GIF image.

제 1 장 서 론

본 논문에서는 CDMA(Code Division Multiple Access) 단말기에서 LZW(Lempel-Ziv-Welch) 데이터 압축 알고리즘을 이용하여 대용량의 이미지를 압축, 복원하여 메모리 효율을 높이는 알고리즘을 설계, 구현하고자 한다.

현재 CDMA 단말기에서 사용되는 이미지는 비트맵(bitmap)으로 메모리에 헥사값으로 저장되어 있다. 이러한 이미지는 표현될 때 각 픽셀(pixel : picture element)을 표시하기 위해 공간과 색상으로 정의된다. 이것은 빠른 데이터 변환과 디스플레이에 적합하므로 흑백 또는 4 그레이(gray) 방식의 이미지들을 사용하는 단말기에서는 유용하게 사용되고 있다[1].

그러나 2000년대 멀티미디어의 시대를 맞이하면서 휴대전화 단말기는 휴대 간편한 단말기, 큰 화면 및 강력한 연산능력을 가진 단말기 등 다양한 형태의 단말기들이 출현하게 되면서 그래픽 측면에서 보다 고품질의 정보 제공이 요구되고 있다. 이제는 256 칼라 또는 65000 칼라 이상의 단말기들이 제조되고 있는데, 이것은 센싱, 신호처리, 통신, 디스플레이 관련 기술과 이들의 기초가 되는 소재 및 반도체 그리고 소프트웨어 기술의 결합에 힘입어 가능하게 된 것이다[2].

좀 더 고급화된 그래픽은 보다 많은 데이터 사이즈를 요구하는데, 실제로 256 칼라의 경우 이미지가 128×160 사이즈일 때 21.5 KB의 데이터가 필요하게 된다. 일종의 운영체제를 가지고 있는 시스템인 단말기로서는 데이터를 저장하는 메모리 영역이 한정적이고, 부품

하나 하나의 단가가 제품의 단가를 결정하고, 판매량에 큰 영향을 미치기 때문에, 이러한 데이터를 그대로 사용한다면 우리가 사용하고 있는 저장매체들의 용량과 통신선로의 대역폭에 비해 너무 큰 데이터일 뿐만 아니라, 경제적으로도 큰 손실을 입게 되므로, 압축 기술을 사용한다면 저장, 처리, 전송에 있어서 많은 이득을 얻게 될 것이다[3].

그러므로, 본 논문에서는 GIF(Graphics Interchange Format) 파일에서 사용하는 LZW라는 압축 알고리즘을 이용하여 이미지를 압축함으로서, 좀 더 효율적인 메모리 사용을 하고자 한다.

GIF 파일에서 추출된 LZW 데이터 영역을 메모리에 저장한 후 단말기의 이미지 프로세싱 과정에서 LZW 데이터를 복원하여 비트맵 데이터와 같이 LCD에 디스플레이 하는 알고리즘을 제안한다.

이 알고리즘을 사용할 경우, 비트맵 파일을 GIF 파일로 변환하였을 때 얻을 수 있는 데이터 압축율을 동일하게 얻을 수 있으므로 단말기에 적용 시 메모리에서 차지하는 데이터 크기의 감소량을 측정하여 효율성 개선을 증명하고자 한다. 본 논문의 구성은 다음과 같다.

제 2 장에서 현재 CDMA 단말기에서 사용하는 이미지 처리 방식과 이미지 압축에 대한 개요를 제시하고 LZW 압축 알고리즘을 휴대전화 단말기에 적용하게된 이유에 대해 설명하였고, 제 3 장에서는 비트맵와 GIF의 구조에 대해서 기술하였고, GIF에서 사용하는 LZW 압축 알고리즘의 압축과 복원 이론 및 구현 방법에 대해서 설명한다.

제 4 장에서는 CDMA 단말기에서 실제 GIF 파일에서 LZW 데이터를 추출하고, 단말기에서 LZW 데이터를 비트맵 데이터로 복원,

시뮬레이션 하는 방법에 대해서 제시하며, 제 5 장에서는 앞서 4 장에서 제시한 이론을 CDMA 단말기에서 사용하였을 경우, 실제 이득이 되는 데이터 사이즈를 측정한다.

제 6 장에서는 결론을 내리고 앞으로의 연구방향을 제시한다.

제 2 장 CDMA 단말기의 이미지 프로세싱

2.1 CDMA 단말기의 이미지 프로세싱 개요

CDMA 단말기에서는 LCD에 표시되는 대부분의 데이터들을 이미지 데이터로 가지고 있다. 단말기가 발달할수록 사용자의 요구에 따라 많은 기능들이 생겨나고, 사용자 인터페이스는 갈수록 많은 양의 그래픽 데이터를 소화하여야만 한다.

현재 CDMA 단말기에서 사용되는 이미지뿐만 아니라 폰트, 심볼 등은 2진 비트맵 형태의 헥사 데이터로 저장되어 있다. 이렇게 비트맵을 사용하는 이유는 비트맵 데이터는 각 픽셀에 대한 정보를 가지고 있기 때문에, 압축 과정을 거치지 않고 즉시 디스플레이 되므로 단말기에 부담을 주지 않고 바로 사용할 수가 있기 때문이다.

그러나 256 칼라 이상의 이미지 데이터가 사용되기 시작하면서, 같은 사이즈의 이미지를 표현하더라도 최소 10배 이상의 데이터가 필요하게 되었다. CDMA 단말기는 REX라는 운영체제를 사용하며 여러 개의 상태를 가지고 있는데, 단말기는 항상 대기화면 상태를 유지하며 사용자의 입력 또는 기지국과의 통신을 대비하고 있기 때문에 시스템에 부하가 걸리는 많은 명령이나 많은 데이터를 호출할 경우 기능을 제대로 수행하지 못하게 된다[1].

만약 CDMA 단말기에서 사용되는 이미지 데이터들을 여러 압축 방식을 이용하여 압축을 하게 된다면, 메모리 사이즈 면에서 많은 이득을 볼 수 있을 것이다.

그러나, 이런 압축 알고리즘을 사용하지 않는데는 이유가 있다.

그 첫 번째 이유는 2 칼라, 4 그레이 데이터를 사용할 때는 그렇게 많은 메모리를 소요하지 않았고, 오히려 이미지의 압축, 복원 과정을 거칠 때 소요되는 이미지의 팔레트, 헤더 데이터, 실제 데이터를 분석하는데 걸리는 소요 시간이 마이너스 효과를 가져오기 때문이었다.

두 번째 이유는 압축 알고리즘에 대한 로얄티를 지불해야 하기 때문으로 들 수 있다. 로얄티는 제품의 단가에 영향을 줄 수 있으므로 치명적이라고 할 수가 있다.

그럼에도 불구하고, 최근 세계적으로 칼라 핸드폰 시장이 확대되면서 데이터를 압축해야 하는 과정은 불가피해졌다. 본 논문에서는 이미지를 압축하면서 시스템에 무리 없이 사용할 수 있는 방법에 대해 논해 보고자 한다.

2.2 이미지 압축 알고리즘에 대한 개요

이미지에는 상당한 양의 공간 중복성이 있다. 따라서 이미지를 압축한다는 것은 이 공간 중복성을 제거하는 것을 말하게 된다. 압축 방법은 이 공간 중복성을 제거하는 방법에 따라 분류된다. 일반적으로 이미지를 압축하기 위한 방법은 크게 두 종류로 분류할 수 있다. 첫 번째가 손실이 없는 압축방법(lossless compression)으로 PCX, GIF, TIFF(Tag Image File Format) 등이 이 부류에 속한다. 이들은 압축했다가 다시 복원한 후의 영상이 원래 영상과 일치하는 반면 압축율이 2:1 이하로 매우 낮다. 정보이론에 의하면 랜덤한 정보를 손실없이 이 이상 압축하는 것은 불가능하다. 이 방법에 기본적으로 사용되는 기술은 예측 부호화(prediction encoding), 실행길이

부호화(run length encoding)와 엔트로피부호화(entrophy encoding) 등이다[3].

이에 비해 손실을 허용하는 압축방법(lossy compression)은 일반적으로 변환 부호화(transform coding) 기술을 사용하여 10:1 또는 수십:1 정도의 압축율을 내는 방법을 말하는데 JPEG(Joint Photographic Experts Group)이 여기에 속한다[3][4].

CDMA 단말기에서는 디스플레이하는 이미지를 원래의 이미지로 그대로 복구해야 하므로 손실이 없는 압축방법을 택해야 했고, 본 논문에서는 그 중 압축과 복원의 속도가 빠르고 문자열로 데이터를 복구해주는 LZW 압축기법을 선택하였다[5].

2.3 LZW를 이용한 압축율 개선 및 처리속도의 증가

ITU-T의 V.42bis에 채용되어 있는 LZW 압축기법은 1978년 이스라엘 Lempel과 Ziv가 제안한 것을 1985년 현재 유니시스사의 전신인 스페리사의 Terry Welch가 수정 구현한 압축기법으로 LZW 방법으로 칭해진다[6].

이 기법은 입력 데이터의 길이를 가변으로 하고 출력부호의 길이를 고정시킨 기법으로서 데이터 압축율이 높으며 내부 연산량이 작기 때문에 압축 수행 속도 측면에서는 현재까지 가장 빠른 것으로 평가되고 있어 ITU-T의 표준 권고를 시점으로 정보통신에 널리 이용될 것으로 보인다.

처음 제안된 LZW 알고리즘은 가변길이의 입력 문자열을 모두 12 비트의 고정 길이로 2진 부호화 했으나 그 후 초기 압축 효율의 개선을 위해 9 비트부터 시작해 문자열 테이블 내에 할당된 부호의 범위에 따

라 12 비트까지 2진 부호화 하도록 개선된 LZW 알고리즘이 주로 사용되고 있다. 현재 LZW 알고리즘을 채용하고 있는 상용 압축파일에는 국내에서 자주 쓰이고 있는 pkarc, pkzip 등이 있다.

LZW 알고리즘에 있어서 문자열 테이블 내에 생성되는 문자열의 갯수는 시스템의 허용능력에 따라 확장할 수 있다. 상용 압축파일인 pkarc는 테이블 내 최대 문자열 엔트리를 4096으로 제한하고 있으며 pkzip은 이를 8192로 제한하고 있다. 일반적으로 허용 가능한 문자열을 늘릴수록 압축율은 향상된다[5].

간단하게 말해서, LZW 압축은 입력되는 텍스트의 아무런 분석 없이 여러 심볼을 묶은 가변길이 심볼열을 가변길이 부호로 표현하는 방식이며, 이는 압축할 자료를 읽자마자 부호화하면서 새로 나오는 심볼열을 사전 식으로 기억시킨 다음에 이 사전에 저장된 자료를 기본으로 부호화에 이용한다. 이런 방식을 'on the fly' 즉, LZW 방법이라고 한다.

제 3 장 비트맵, GIF와 LZW의 개요

3.1 비트맵의 구조

비트맵은 본래 마이크로소프트가 윈도우 사용자들을 위해 개발한 고유의 그래픽 파일 형식이다. 윈도우가 기동하거나 종료될 때 보여지는 이미지들이나 바탕화면을 꾸며주는 배경 그림 등은 모두 비트맵 파일 형식이다. 비트맵은 그림 데이터를 비효율적으로 저장함으로써, 필요 이상으로 파일 크기가 커지는 경향이 있다. 그러나 적어도 배경그림으로 사용되는 파일의 크기만은 실행길이 부호화라고 불리는 기술을 사용하여 크기를 줄일 수 있다.

비트맵은 작은 점들의 2차원 배열을 이용하여 영상정보를 표현하는 방식을 말하며, 이렇게 그림을 이루는 작은 점들을 픽셀이라 부르며, 각 점들에 표현되는 칼라에 의해 그림이 이루어지게 된다[7].

3.1.1 비트맵의 종류

① DDB(Device Dependent Bitmap)

디바이스에 종속적인 비트맵을 말한다. 여기서 디바이스 종속이라는 것은 비트맵의 한 픽셀이 몇 비트로 표시될 것인지 제어판의 디스플레이 등록정보에 설정되어 있는 화면의 설정에 종속된다는 의미이다. 화면에 출력되기 위해 디바이스에 선택된 비트맵이나, 디바이스와 호환성을 갖도록 만들어진 메모리 비트맵을 DDB라 한다.

② DIB(Device Independent Bitmap)

DIB는 현재 시스템의 디스플레이 등록정보와 무관하게 나름대로 생각을 표현하는 비트맵을 DIB라고 한다. 일반적으로 그래픽 파일의 형태로 저장되어 있는 비트맵은 대부분 DIB이며, 흑백 그림파일은 어떤 컴퓨터에서 보든 흑백으로 나오고, 칼라 그림은 어떤 컴퓨터에서 보든 칼라로 나오게 된다.

3.1.2 비트맵의 칼라 모드

① 트루 칼라 이미지

이미지의 각 픽셀을 빨간색(R), 녹색(G), 파란색(B) 1 바이트씩의 조합인 24 비트를 사용함으로써 차지하는 메모리 양은 커지지만 매 픽셀을 1,600만 가지의 색으로 표현할 수 있기에, 최상급의 화질로 이미지를 저장할 수 있다.

② 인덱스 칼라 이미지

트루 칼라 이미지는 용량이 매우 커지기 때문에 화질이 약간 떨어지더라도 용량을 줄일 필요가 있다. 실제로 픽셀 당 8 비트를 할당하면 사람의 눈으로 보기에 충분히 깨끗한 화질을 얻을 수 있다. 따라서, 그림 3-1과 같이 이미지에서 많이 사용되는 256(8 비트)개의 색을 추려서 사용하면 이미지의 용량을 1/3로 줄일 수 있게 된다. 이때 1600가지 색 중에서 이미지에 사용된 색 256가지는 그림 3-2와 같이 팔레트라는 표로 만들어 저장되어 있으며, 각 픽셀의 값은 색을 표현하는 값이 아니라 팔레트의 인덱스 값을 가지게 되는 것이다. 물론 팔레트가 256개의 색을 가지고 있을 필요는 없으며

64, 16가지 등의 팔레트를 가질 수도 있다. 이때 각 픽셀은 6 비트, 4 비트의 메모리만 사용하게 되지만, 이미지의 화질이 상대적으로 낮아진다는 단점이 있다.

255	12	50	80	123	221
30	50	80	67	130	145
56	80	158	130	145	175
80	10	130	145	175	200
89	130	145	175	29	182
113	144	135	200	22	45

그림 3-1. 인덱스 비트맵 이미지

index	R	G	B
255	·	·	·
254	·	·	·
253	·	·	·
…	·	·	·
·	·	·	·
1	·	·	·

그림 3-2. 인덱스 칼라 팔레트

팔레트에 담겨있는 색들이 모두 회색 계열일 경우, 흑백 이미지 또는 그레이 스케일 이미지라고 한다. 다음 그림 3-3와 같이 RGB

가 동일한 비율로 혼합되면 회색이 되므로, 흑백 이미지에서는 인덱스 값이 곧 밝기 값이 되며, 각각의 값이 고정되어 있어 실제로 팔레트는 불필요하게 된다.

index	R	G	B
255	255	255	255
254	254	254	254
253	253	253	253
…	.	.	.
2	2	2	2
1	1	1	1
0	0	0	0

그림 3-3. 흑백 스케일 이미지의 팔레트

2진 이미지의 경우에는 흑, 백으로만 표현하는 이미지이며, 픽셀당 1 비트를 사용한다. 팩스 또는 문서 등에서 사용하고 있으며, 서론에서 언급했듯이 CDMA 단말기에서도 많이 사용되었다.

3.1.3 비트맵의 파일 포맷

그림 3-4은 가장 간단한 파일 포맷으로 윈도우에서 사용하는 표준 DIB 파일 포맷이다. DIB 포맷으로 구성된 비트맵 파일은 비트맵 파일에 대한 정보, 비트맵 자체에 대한 정보, 팔레트, 이미지 비트 데이터로 구성된다.

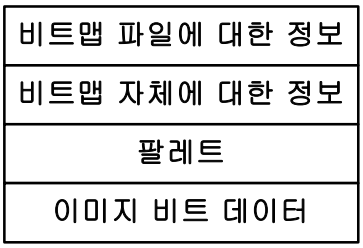

그림 3-4. 비트맵 파일 포맷

① 비트맵 파일에 대한 정보

비트맵값 저장, 파일의 크기, 예약된 값 2가지, 실제 이미지, 비트까지의 옵셋 정보가 들어 있다.

② 비트맵 자체에 대한 정보

구조체의 크기, 이미지의 폭(픽셀 단위), 이미지의 높이(픽셀 단위), 비트 플레인 수, 픽셀 당 비트의 수, 압축 유형, 이미지의 크기(바이트 단위), 가로 해상도, 세로 해상도, 실제 사용되는 색상의 수, 중요한 색상 인덱스가 저장되어 있다.

③ 팔레트

트루 칼라인 경우에는 필요 없으며, 그 이외의 경우 색상 종류만큼의 RGB(24 비트) 색상이 저장되어 있다.

④ 이미지 비트 데이터

실제 비트맵 데이터 상으로는 메모리 내부에서 이미지의 아래 부분부터 저장하게 된다. 다시 말해서 실제 메모리 상의 그림은 역상이 된다.

3.2 GIF의 구조

GIF는 JPEG과 함께 월드 와이드 웹(WWW : World Wide Web)에서 지원되는 두 가지 그래픽 파일 형식 중의 하나이다. 웹이나 기타 인터넷의 다른 서비스에서 GIF는 이미지 형식에 있어서 사실상의 표준이 되어가고 있다.

실제로 이 형식에 대한 권리는 컴퓨서브(Compuserve)에 의해 소유되고 있으므로, GIF 포맷을 이용해 제품을 만들고자 하는 회사는 라이센스를 받아야만 한다. 그렇지만 웹 페이지에 GIF 이미지들을 포함시키는 일반 사용자나 회사의 경우는 예외이다.

기술적인 측면을 보면 GIF는 2차원 래스터 데이터(raster data) 형식으로 바이너리로 표현되며, 압축기술은 LZW를 이용한다. GIF에는 87a와 89a의 두 가지 버전이 있는데, 1989년 7월에 발표된 89a는 하나의 파일 내에 짧은 순서를 갖는 일련의 이미지들을 포함시킴으로써 애니메이션 GIF를 만들 수 있는 기능과, 인터레이스(interlaced GIF) 표현 기법을 지원한다[7][8][9]. 그림 3-5는 GIF 파일 포맷을 나타낸다.

① GIF 서명

GIF 서명은 데이터가 적절한 GIF 이미지 포맷으로 이루어졌다는 것을 말해준다. 이것은 'G I F' 라는 3 바이트의 문자와 3 바이트의 버전으로 구성되어 있다. GIF로 간주되는 문서를 참조할 때 일반적으로 사용된다.

그림 3-5. GIF 파일 포맷

② 스크린 디스크립터

스크린 디스크립터는 GIF 이미지에 따르는 파라미터들을 포함하고 있는데, 이것은 이미지 전체의 넓이 또는 로컬 스크린에서 요구하는 넓이, 칼라 맵핑 정보의 유무, 백그라운드 스크린 칼라, 칼라 깊이 정보 등을 말해 준다. 이 정보들은 다음 그림 3-6와 같이 8비트 바이트의 순서로 저장되어 있다.

논리적인 스크린의 넓이와 높이는 실제 디스플레이 되는 넓이보다 크다. 실제 디스플레이보다 이미지가 더 큰 이유는 구현 또는 하드웨어적인 특징에 따라 틀리다. 만약 그렇지 않다면, 이미지는

디스플레이 될 때 가장자리가 잘려 나갈 것이다.

픽셀의 값은 표현 가능한 최대 색의 수를 뜻한다. 픽셀값의 범위는 1-8 비트를 말해주는 0-7 사이이며, 이것은 2 칼라 (흑 또는 백)에서 256 칼라까지 변환된다는 것을 뜻한다.

그림 3-6. 스크린 디스크립터의 구조

③ 글로벌 칼라 맵

글로벌 칼라 맵은 선택적이나 정확한 칼라의 표현을 위해서 권고되고 있다. 이 칼라 맵이 존재한다는 것은 스크린 디스크립터에 5 바이트의 'M' 필드가 존재한다는 것을 말해준다. 이 글로벌 맵은 보통 장비상 하드웨어의 제한 때문에 사용되었고, 개인적인 이미지 디스크립터에서 'M' 필드는 보통 0으로 명시되어 있다. 만약 글로벌 이미지 디스크립터가 존재한다면, 이것은 스크린 디스크립터에 따를 것이다.

그림 3-7과 같이 칼라 맵 요소의 개수는 스크린 디스크립터에 따를 것이며, 빨간 색, 초록색, 파란색 색깔의 세기를 표현하는 3 바이트 값으로 구성되어 있다.

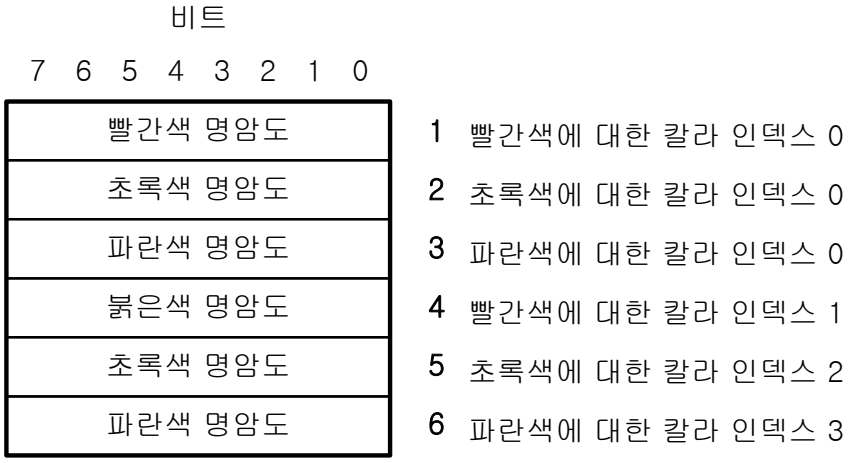

그림 3-7. 글로벌 칼라 맵

각 이미지의 픽셀들은 칼라 맵에서 가능한 가장 유사한 색상에 맞추어 디스플레이 된다. 칼라 컴포넌트는 0에서 255까지의 부분적인 세기를 보여준다. 흰색은 (255, 255, 255), 검은 색은 (0, 0, 0)으로 중간 노랑 색은 (180, 180, 0)으로 표현된다.

만약 이미지를 표현하는데, 하드웨어가 8 비트 칼라 보다 적은 이미지를 지원한다면 8 비트 포맷으로 변환시켜야 하는데, 다음 그림 3-8의 공식에 따른다.

```
<map_value> = <Component_value> x 255 / (2** <nbits> - 1 )
```

그림 3-8. 8 비트 포맷으로 바꾸기 위한 공식

이것은 만약 하드웨어에서 칼라 팔레트 없이 이미지를 만들 경우, 하드웨어에서 표현 가능한 색상으로 이미지를 생성했을 경우, 또는

글로벌 칼라 맵이 없을 경우 내부적으로 하드웨어에서 표현 가능한 색으로 칼라맵을 생성한다.

④ 이미지 디스크립터

이미지 디스크립터는 그림 3-9와 같이 실제 이미지의 배치와 이미지의 넓이, 로컬 칼라 룩업(lookup) 맵 유무를 의미하는 플래그, 연속적으로 디스플레이 되는 픽셀을 정의한다. 이미지 디스크립터에는 도입을 알리는 이미지를 분리하는 문자가 있는데, 이것은 ' , ' 또는 '0x2C'로 표현된다.

그림 3-9. 이미지 디스크립터

⑤ 로컬 칼라 맵

로컬 칼라 맵은 선택적이며, 차후의 용도를 위해 정의되었다. 만

약 이미지 디스크립터에서 'M' 비트가 부과되어 있다면, 칼라 맵은 이미지에 적용시킬 때 이미지 디스크립터를 따른다.

⑥ 래스터 데이터

실제 이미지 화소의 값을 나타내는 영역이다. 이미지의 포맷은 픽셀 색상에 대한 인덱스 값의 일련으로 정의된다. 픽셀은 이미지 열에서 왼쪽에서 오른쪽으로 연속되게 저장되고, 각 이미지의 열은 위에서 아래로 연속되게 쓰여진다.

3.3 LZW 압축 알고리즘의 구조 및 구현

LZW는 문자 데이터의 반복성을 이용한 데이터 압축 방식이다. 래스터 데이터가 보통 많은 반복성을 가지기 때문에, LWZ는 이 데이터를 압축, 압축복원 하기에 좋은 방법이 된다. 일반적인 LZW의 압축과 복원에 관해서 알아보면, GIF의 변화량은 LZW로부터 나온 것을 알수 있다.

LZW는 압축과 압축복원에서 3가지 요소를 다루고 있는데, 그 3가지는 문자열, 코드열, 문자열 표로 구성되어 있다. 압축 과정에서는 문자열이 입력, 코드열이 출력되며, 압축복원 과정에선 코드열이 입력, 문자열이 출력된다[8].

LZW 압축시에 가장 먼저 해야 할 것은 문자열 표을 초기화시키는 것이다. 초기화하기 위해서는 코드 사이즈, 문자의 개수를 알아야한다. 예로 코드 사이즈가 12 비트라면 0부터 FFF만큼, 즉 4096개의 요소를 문자열 표에 저장할 수 있다는 것을 의미한다[6][10].

이 장에서는 압축과 압축복원 알고리즘에 대해 설명하고 있는데,

알고리즘에 있어서 다음과 같은 몇 가지 요소들이 필요하다.

① 문자 : 가장 기본적인 요소로 일반적인 텍스트 파일에서 이것은 1 바이트이다. 래스터 이미지에서는 주어진 픽셀의 색깔을 나타내는 인덱스이다. 여기에서는 'K'라고 칭한다.
② 문자 스트림 : 데이터 파일에서의 문자의 연속.
③ 스트링 : 연속되는 문자들. 여기에서는 [. . .]K라고 칭한다.
④ 접두사 : 이전의 문자를 뜻한다. 문자와 합쳐서 문자열을 이룬다. 여기서에서는 [. . .]로 칭한다.
⑤ 코드 : 스트링으로 매칭시켜야 할 숫자이다.
⑥ 코드 스트림 : 출력되는 코드의 열이다. 래스터이다.
⑦ 문자열 표 : 코드와 매칭되는 스트링을 표현해 놓은 표이다.

3.3.1 LZW 압축

```
① Initailize string table;
② [.c.] <-empty
③ K <- next character in charstream
④ Is [.c.]K in string table?
   ( Yes : [.c.] <- [.c.]K;
           go to [3]; )
   ( No : Add [.c.]K to the string table;
          output the code for [.c.] to the codestream;
          [.c.] <- K;
          go to [3]; )
```

그림 3-10. LZW 압축 알고리즘

표 3-1. LZW 압축 과정

단계	위치	문자열 표	출력
1	1	AB	#1
2	2	BB	#2
3	3	BA	#2
4	4	ABA	#4
5	6	ABAC	#7
6	-	-	#3

위의 그림 3-10 압축 알고리즘을 살펴보면, 문자열을 압축할 때, [.c.]에 문자를 순차적으로 받아들이면서 해당 문자가 문자열 표에 존재한다면, 그 값을 코드열으로 출력하면서 다음 문자와의 연산을 문자열 표에 등록시키는 것을 보여준다.

예를 들어 표 3-1과 같이, 'ABBABABAC'라고 제시된 문자열을 압축한다면, 위의 표 3-1과 같은 압축 과정을 거치게 된다.

첫 번째 문자가 'A'이므로 [.c.]은 'A'가 된다. 'A'는 테이블에 존재함으로 다음 문자와 연산하게 되고, 그 값인 'AB'가 문자열 표에 없음으로 #1을 출력하게 되고 'AB'를 문자열 표에 추가시킨 다음 [.c.]에 'B'가 입력되게 된다. 'B'는 문자열 표에 존재함으로 다음 문자와 연산하게 되고, 그 값인 'BB'가 문자열 표에 없음으로 #2를 출력하게 되고 'BB'를 문자열 표에 추가시킨 다음 [.c.]에 'B'가 입력되게 된다. 이와 같은 방식으로 연산해 갈 경우 단계 4에서는 단계 1에서 추가시킨 'AB'를 #4으로 출력하게되고 단계 5에서는 단계

4에서 추가시킨 'ABA'를 #7으로 출력하게된다. 결국 코드열은 '#1#2#2#4#7#3'이 되게 된다.

효과적으로 압축을 하기 위한 중요한 사항들이 몇 가지 있는데, 그 첫 번째로는 해싱 알고리즘을 사용하는 것이다. 문자열 표을 검색하는데 있어서 해싱 알고리즘은 정확하게 계산을 수행할 것이다. 또한 "Straight LZW" 압축을 사용함에 있어서 문자열 표의 오버플로우의 위험성을 알아야 한다. 위와 같은 문제점들을 해결하기 위한 몇 가지 방법이 있다[11].

또, 중요한 한 가지로는 압축과정에 있어서 우리가 [. . .]K를 문자열 표에 가지고 있다면, [. . .] 역시 문자열 표에 있다는 것을 알 수 있다. 이것은 곧 우리가 문자열의 전체를 저장하고 있다면 어떤 문자열이든 전 문자와 문자의 합으로 표현된다는 것을 의미한다. 그러므로 우리가 [. . .]K를 문자열 표에 가지고 있다는 것 자체가, [. . .]을 이미 알고 있다는 것을 의미한다[12][13][14].

3.3.2 LZW 복원

그림 3-11 복원 알고리즘을 살펴보면, 먼저 문자열 표를 초기화시킨다. 이 표에 관한 정보는 문자열로부터 온다. GIF 파일의 경우는 몇 개의 표현 가능한 픽셀 값이 존재하는지에 대한 정보를 헤더로부터 얻어 온다. 그리고, 코드열을 압축 해제하면서 문자열 표를 완성시킬 것이다.

먼저 압축을 풀기 위해서는 현재 입력되는 코드가 필요하다. 이것은 <code>, 입력되었던 코드는 <old>로 칭해진다. 처음 코드를 <code>에 입력하고 문자열 표를 검색해, 그에 해당되는 값을 문자

열에 입력하고, <code>를 <old-code>로 만든다. 그리고 그 다음 코드를 <code>에 입력하고 만약 이 코드가 문자열 표에 존재한다면 그에 해당되는 값을 문자열에 입력하고, <old-code>와 <code>의 합을 문자열 표에 추가시킨다. 그러나, 입력받은 코드가 문자열 표에 존재하지 않는다면, <old-code>와 스트링의 처음 문자를 찾아서 합친 다음 문자열 표에 추가시킨다.

```
① Initialize string table
② get first code: <code>;
③ output the string for <code> to the charstream;
④ <old> = <code>;
⑤ <code> <- next code in codestream;
⑥ does <code> exist in the string table?
   ( Yes : output the string for <code> to the charstream;
           [. . . ] ← translation for <old>;
           K ← first character of translation for <code>;
           add [. . . ]K to the string table;
           <old> ← <code>; )
   ( No : [. . . ] ← translation for <old>;
          K ← first character of [. . . ];
          output [. . . ]K to charstream and add it to string table;
           <old> ← <code>; )
⑦ go to ⑤;
```

그림 3-11. LZW 복원 알고리즘

즉, 예를 들자면 위의 압축에서 나온 코드열을 다시 문자열으로 변환시키고자 할 때 코드열은 '#1#2#2#4#7#3'이다.

표 3-2. LZW 복원 과정

단계	코드	출력	문자열 표
1	#1	A	
2	#2	B	#4 = AB
3	#2	B	#5 = BB
4	#4	AB	#6 = BA
5	#7	ABA	#7 = ABA
6	#3	C	#8 = ABAC

위의 표 3-2의 과정을 보면 <code>는 #1을 입력받고 해당 값이 문자열 표에 존재함으로 'A'를 출력하면서 <old-code>에 'A'를 입력한다. 다음 코드인 #2를 <code>에 입력받고 #2가 문자열 표에 존재함으로 'B'를 출력한 다음 <old-code>와 <code>의 합 'AB'를 문자열 표에 #4로 추가시킨다.

이런 식으로, 연산해나갈 경우, 4번째 코드에서 단계 2에서 추가시킨 #4를 'AB'로 바로 출력하게 된다. 그러나, 단계 5의 경우 #7은 문자열에 존재하지 않는 코드이다. 이런 경우에는 <old-code>와 코드열의 첫 번째 코드를 합하여 'ABA'를 출력하고 문자열 표에 #7로 추가시킨다. 단계 6까지의 연산을 마치면 출력된 코드는 'ABBABABAC'와 같이 되어 표 3-1의 LZW 압축과정에서 들었던 예의 문자열 입력 값과 같아짐을 확인할 수 있다[12][13][14].

제 4 장 개선된 이미지 알고리즘의 구현

4.1 GIF 이미지에서 LZW 데이터 추출 알고리즘

CDMA 단말기는 해당 이미지의 LZW 데이터를 가지고 있어 디스플레이 할 경우에 이미지를 복원하여 보여준다. 이 LZW 데이터는 미리 GIF 이미지에서 추출하여 메모리에 저장하여야 한다. GIF 파일 자체를 가지고 있어도 상관없지만, 불필요한 헤더 정보들을 포함하게 되고 복원 시에 헤더와 래스터 데이터를 분리해야 하는 불필요한 작업을 수행해야 하기 때문이다.

그림 4-1은 GIF 파일을 로딩하면서 불필요한 GIF 파일의 헤더를 추출하고 필요한 래스터 데이터만 저장하는 것을 보여준다.

헤더는 앞서 3장의 GIF의 개요에서 설명했듯이, 첫 번째 6개의 문자에서 버전을 추출해내고, GIF 파일이 맞다면, 스크린 디스크립터 부분에 해당하는 글로벌 칼라 테이블의 수, 칼라의 수, 글로벌 칼라 맵, 백그라운드 칼라, 팔레트를 추출할 수 있다.

그리고, 이미지 디스크립터에서는 미지의 좌측과 상단의 위치, 이미지의 가로와 높이를 추출하고, 로컬 칼라 맵에서 로컬 칼라 테이블의 수, 인터레이스 모드, 로컬 칼라 맵을 추출한다. 그리고, 실제 LZW 데이터 포함하고 있는 래스터 데이터를 파일의 마지막 부분까지 읽으면서 버퍼에 저장하게 된다.

이와 같이 저장된 래스터 데이터 영역을 C 파일로 저장하게 되고, 디스플레이 시에 복원하여 보여주게 되는 것이다[15].

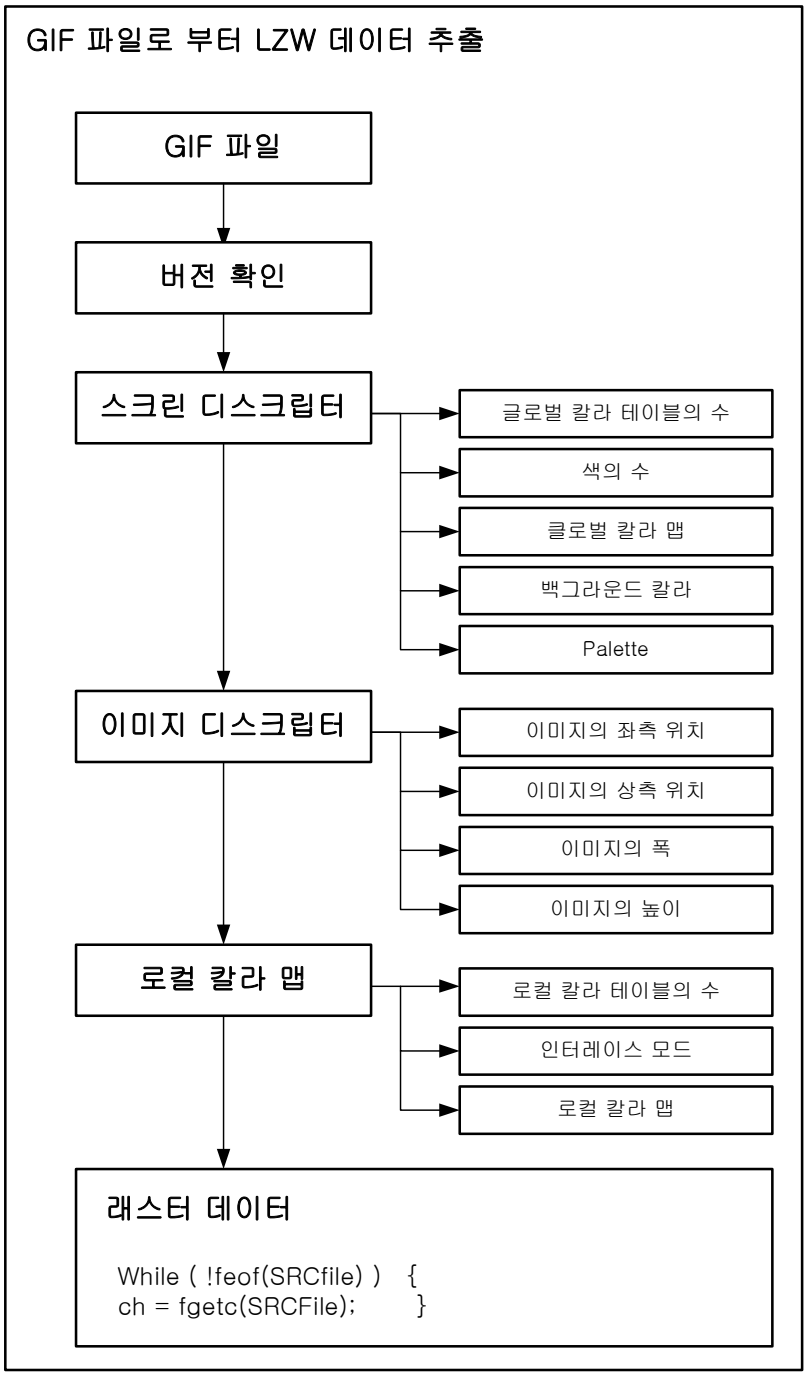

그림 4-1. GIF에서 LZW 데이터 추출 알고리즘

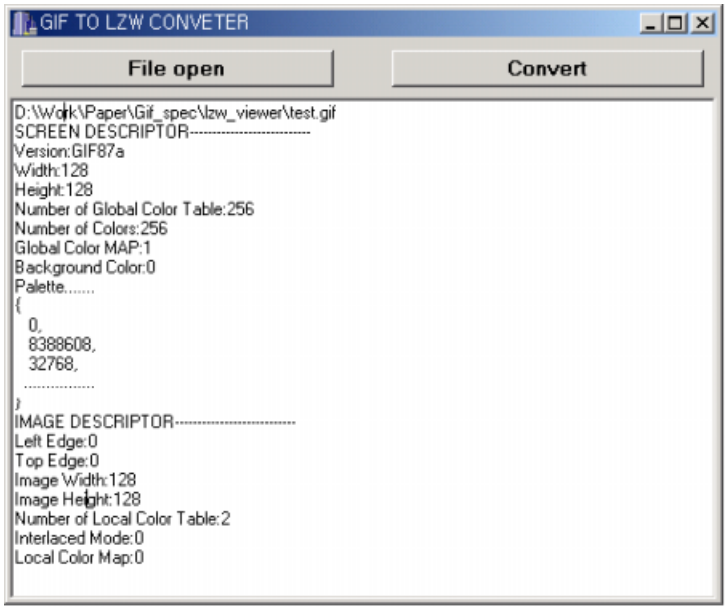

그림 4-2. GIF 파일로부터 LZW 데이터 추출

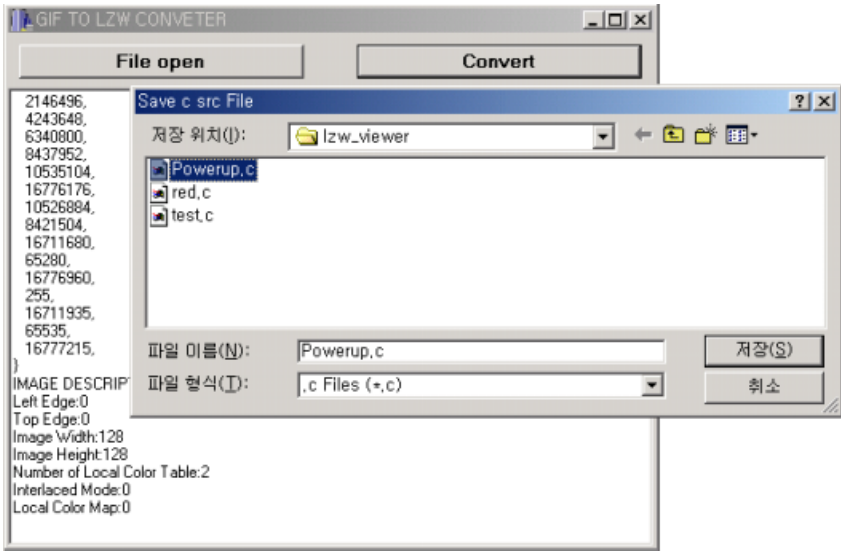

그림 4-3. C 파일로 저장되는 LZW 데이터

그림 4-1의 알고리즘대로 프로그램을 구현해 본 결과 이미지의 헤더에 들어가는 버전, 스크린 디스크립터, 이미지 디스크립터 등을 그림 4-2의 프로그램의 인터페이스에서 확인할 수 있다.

래스터 데이터를 다음 그림 4-3와 같이 단말기 소스에서 사용할 수 있도록 C 파일로 저장을 하면 그림 4-4와 같이 저장된다.

```
char  Powerup[]=
{ //Powerup.c W=128,H=128
0x80,0x80,
0x08,0xff,0x00,0x77,0x08,0x1c,0x48,0xb0,0xa0,0xc1,0x83,
0x70,0x86,0x43,0x87,0x32,0x22,0x4a,0x9c,0x28,0x51,0x85,
0x48,0x91,0x62,0x23,0xc7,0x8f,0x20,0x43,0x32,0x1c,0x49,
0x21,0x52,0x5c,0xa9,0xb1,0xa5,0xc5,0x90,0x30,0x63,0x82,
0x9a,0x3b,0x6a,0xde,0xcb,0xf9,0x6f,0xa7,0x4e,0x9e,0x3b,
0x25,0x4b,0x97,0x18,0x65,0x72,0xbc,0xc9,0x94,0x29,0x47,
0x27,0x45,0x67,0xd5,0x9e,0x57,0xef,0x65,0x95,0x32,0xb4,
0x4e,0x44,0x9a,0x54,0xe9,0x52,0xa9,0x57,0x9b,0x52,0x4d,
0x5b,0xad,0xca,0xc5,0xea,0xb5,0x6e,0x51,0xb1,0x11,0xc9,
0x52,0xb5,0xff,0x92,0xb0,0x6d,0x4b,0x73,0x07,0xd7,0xb9,
0xf4,0xae,0x58,0xbd,0x1e,0xf9,0xfe,0xf5,0x9b,0x02,0x70,
0xc4,0xaa,0xd8,0x2a,0x63,0xa1,0x8e,0x8d,0x92,0x95,0x0c,
0x2d,0x0b,0x2e,0x58,0x73,0x31,0xd0,0xb8,0x9c,0xe9,0x7e,
0xd2,0x1f,0x51,0x87,0x30,0xad,0xdb,0x9e,0xef,0xdf,0xbe,
0x73,0xe2,0xd8,0xc7,0xb5,0xce,0x1e,0x19,0xfa,0x68,0x4b,
0x7a,0x1a,0xf5,0x81,0xeb,0xd8,0xaf,0x0b,0x27,0xee,0xba,
0xd6,0xff,0x7e,0x8c,0x94,0xaf,0xf4,0xf3,0xba,0xa9,0xea,
```

그림 4-4. 헥사 데이터로 저장된 LZW 데이터의 배열

그러나, LZW 데이터를 추출하기 전에 주의해야할 점은 사용할 LCD의 규격에 따른 팔레트이다. 비트맵 이미지을 제작하는 프로그램의 팔레트와 LCD의 팔레트가 다른 경우에는 이미지가 깨져 보이기 때문이다.

현재 개발중인 단말기의 메인 LCD의 사양은 표 4-1과 같다.

표 4-1. 메인 LCD 규격서

컨트롤러	S-43200A
크기	101 x 80 x 8
타입	256 칼라
제조사	Seiko
제조 넘버	S-43200A

위의 LCD에서는 256 칼라의 이미지를 사용한다. 그리고, 3.1절 비트맵의 구조에서 보면 알 수 있듯이 256 칼라의 비트맵 이미지는 256개의 색상을 가진 팔레트를 가지고 있다. 그러나, 밑의 그림 4-5과 같이 두 팔레트는 동일하지 않다.

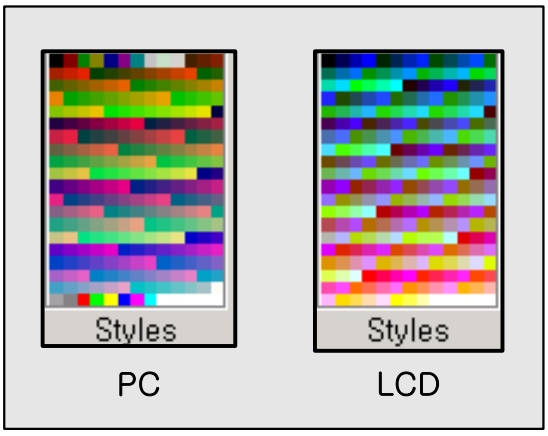

그림 4-5. PC와 LCD의 팔레트 비교

일반적인 PC의 팔레트는 그림 4-5의 왼쪽의 팔레트와 같다. 그러므로, PC의 팔레트로 표현된 이미지의 경우에는 실제 LCD에서는 동일한 색상으로 표현되지 않는다. 그러므로, LZW 데이터를 GIF 이미지에서 추출하기 전에 LCD에 맞는 팔레트로 표현된 이미지를

구현해야한다[16].

4.2 단말기에서 LZW 데이터 복원 알고리즘

4.1절에서는 GIF 이미지 파일에서 LZW 데이터를 추출하여 C 파일에 저장하도록 하였다. 이 장에서는 그림 4-6와 같은 문자열을 복원하여 비트맵과 같이 LCD에 표현하는 방법에 대해서 구현하였다.

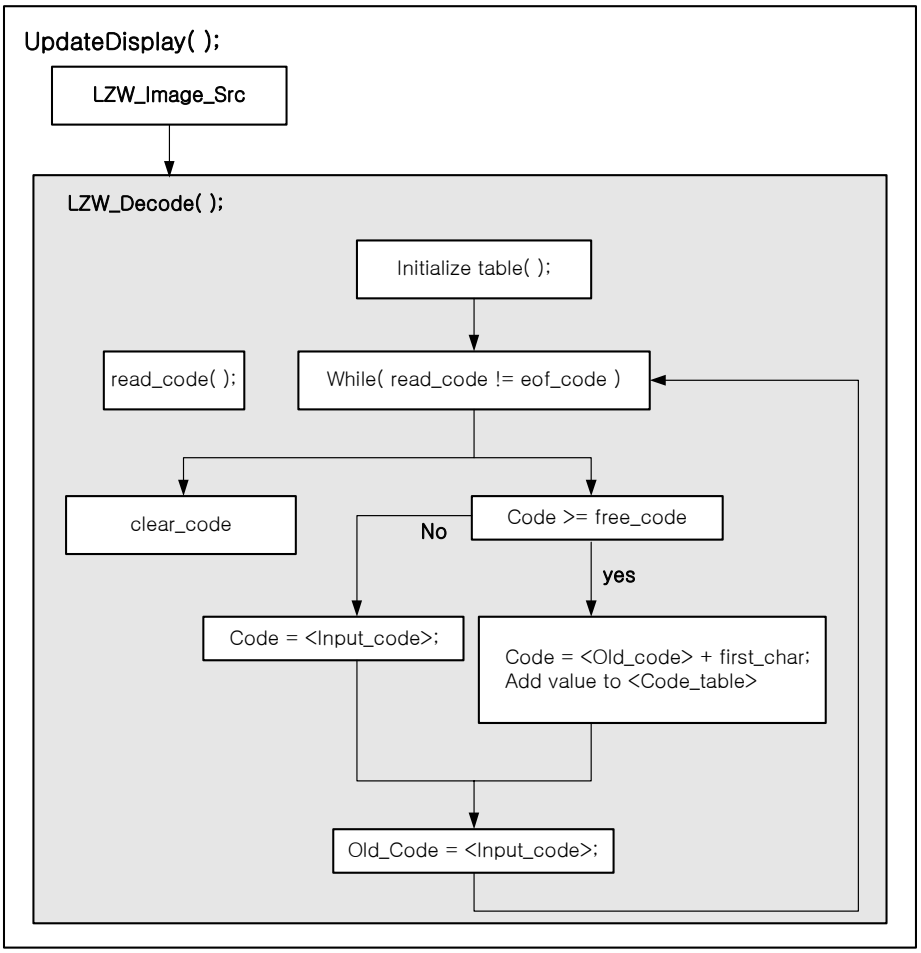

그림 4-6. LZW 압축 데이터 복원 알고리즘

단말기의 LCD_Display 관련 부분 중 화면을 실시간적으로 업데이트 해주는 함수인 UpdateDisplay( ) 함수는 위의 LZW_Decode( ) 함수를 호출하고 있다. LZW_Decode 함수에서 입력된 LZW 데이터를 복원하여서 UpdateDisplay( ) 함수로 리턴하게 되고, UpdateDisplay( )에서는 리턴된 데이터를 디스플레이 하도록 한다. LZW_Decode( ) 함수에서는 입력된 LZW_Image_Src를 read_code( ) 함수가 소스 데이터를 순차적으로 읽어와 그 코드가 Code_table에 존재한다면 문자열 값으로 바꾸어 출력해주고 그 코드가 Code_table에 존재하지 않는다면 <Old_code>와 First_char의 합을 Code_table에 등록시키고 문자열으로 바꾸어 출력 하도록 한다. 그리고, <Old_code>에 출력한 값을 입력한다.

위와 같이 복원한 값들은 비트맵 이미지의 헥사값과 동일하게 되고, 똑같은 디스플레이 효과를 가지게 된다.

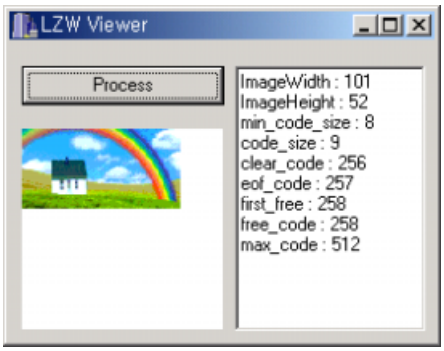

그림 4-7. LZW 뷰어에서 LZW 데이터 디스플레이

위의 그림 4-7은 LZW_Decode( )를 사용하여 윈도우 상에서 제대로 디스플레이 되는지 알아본 것이다.

그림 4-8. LCD 에뮬레이터에서 LZW 데이터 디스플레이

위 그림 4-8은 비트맵 이미지를 GIF로 변환 시켜, 4.1절에서 제시하였던 GIF TO LZW라는 프로그램을 이용하여, LZW 데이터를 추출하고, 추출한 LZW 데이터를 포함한 소스를 컴파일 한 후 단말기에 다운로드하고, 시리얼 케이블로 단말기와 PC를 연결하여 에뮬레이터를 통하여 대기화면을 확인한 모습이다.

다음 그림 4-9와 같이 실제 PC에서 만든 256 칼라 비트맵과 GIF 이미지와 위의 그림 4-8의 에뮬레이터 이미지를 비교하였을 경우 똑같이 디스플레이 되었음을 알 수 있다.

그림 4-9. 대기화면의 비트맵 이미지와 GIF 이미지 비교

결국 비트맵과 똑같은 이미지를 디스플레이 하더라도 위의 과정을 거친다면 더욱 작은 용량의 데이터로 같은 이미지를 디스플레이 할 수 있는 것이다.

제 5 장 실험 및 검증

5.1 LZW 알고리즘 적용시의 메모리 사이즈 비교

칼라 LCD를 사용하게 되면서 단말기에서는 많은 이미지들을 사용하게 되었다. 칼라 LCD에 맞추어 사용자 인터페이스에는 다양한 기능들이 생겨나게 되고 기본적인 대기화면, 수신 화면, 송신 화면, 배터리, 달력, 세계지도, 다이어리 등의 수많은 이미지들이 사용되고 이에 소요되는 메모리 사이즈는 급격히 증가하게 되었다.

현재 개발하고 있는 단말기에 들어가는 이미지의 개수는 대략 273개 정도이다. 대기화면의 경우에는 다음 그림 5-1과 같이 대기화면 하나를 표현하는데도 10 프레임 정도의 이미지가 필요하기 때문에, 기능이 늘어날 때마다 이미지의 수는 급격하게 늘어나고 있다.

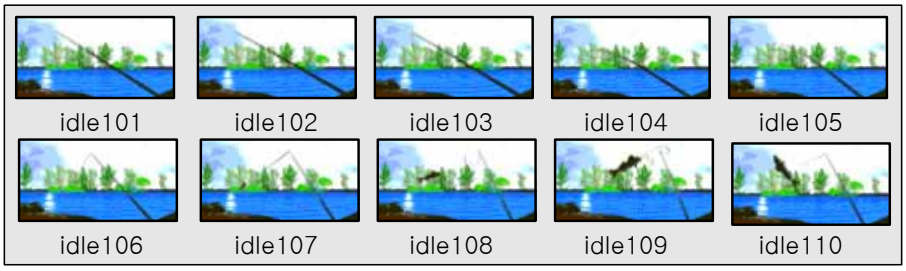

그림 5-1. 대기화면 ①의 프레임

LZW 데이터 복원을 통해서 얻을 수 있는 압축율은 이미지의 색상, 구성에 따라서 랜덤하게 틀려진다. 단색의 101×52 크기의 이미지의 경우에는 121 바이트 밖에 사용되지 않는 것이다.

그래서, 단말기에서의 실제적인 효율을 알아보기 위해서 가장 많이 사용되는 대기화면 경우에는 어느 정도의 압축율을 보이는지 알아보았다.

그림 5-1의 101×52 크기의 대기화면의 경우 그림에 따라 틀리지만 한 개의 이미지를 표현하기 위해서 현재 소요되는 메모리 사이즈는 대략 2000 바이트 이상의 메모리가 소요된다. 비트맵을 사용하였을 경우와 비교해 1/3정도의 메모리를 소요하고 있는 것이다. 현재 단말기에 사용되고 있는 이미지의 총 사이즈를 합산하였을 경우에는 비트맵을 사용하였을 때 보다 실로 많은 양의 메모리를 이득을 얻게 된다.

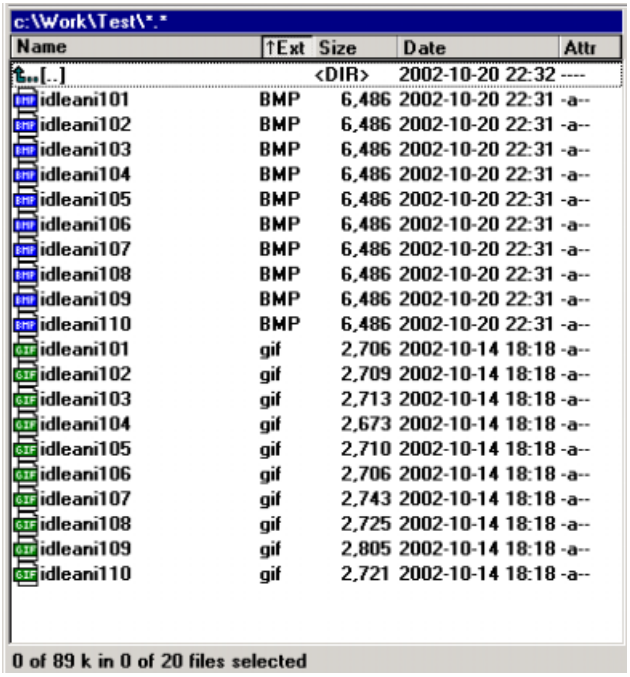

그림 5-2. 비트맵 파일과 GIF 파일의 사이즈 비교

그림 5-2는 그림 5-1의 대기화면 GIF 파일의 크기와 비트맵 파일 크기 비교한 것으로 대략 41% 내외의 크기가 차이가 있음을 알 수 있다.

그리고, 위의 데이터에서 실제 단말기에서 비트맵의 래스터 데이터 부분을 추출해서 사용할 경우와 GIF 이미지에서 LZW 데이터 부분을 추출하여 사용하였을 경우 이미지 디스플레이 시에 실제적으로 얻을 수 있는 압축율을 다음 표 5-1에 비교해 보았다.

표 5-1. 대기화면 ①의 데이터 압축율 비교표

(그림 5-1의 대기화면, 101×52 크기 이미지 (바이트 단위))

파일 이름	비트맵 이미지			GIF 이미지			압축율 (%)
	파일 크기	헤더	데이터	파일 크기	헤더	데이터	
Idle101	6,486	1,078	5,408	2,706	791	1,915	65%
Idle102	6,486	1,078	5,408	2,709	791	1,918	65%
Idle103	6,486	1,078	5,408	2,713	791	1,922	64%
Idle104	6,486	1,078	5,408	2,673	791	1,882	65%
Idle105	6,486	1,078	5,408	2,710	791	1,919	65%
Idle106	6,486	1,078	5,408	2,706	791	1,915	65%
Idle107	6,486	1,078	5,408	2,743	791	1,952	64%
Idle108	6,486	1,078	5,408	2,725	791	1,934	64%
Idle109	6,486	1,078	5,408	2,805	791	2,014	63%
Idle110	6,486	1,078	5,408	2,721	791	1,930	64%

위의 표 5-1을 보면 101×52 크기의 이미지에서는 비트맵의 경우는 비트맵의 구조에 따라 일정한 데이터 사이즈인 5,408 바이트를 보였으나 LZW 데이터일 경우에는 63%-65%까지의 대략 1/3 정도의 데이터를 유지했다. 그러나, 위의 대기화면 ①의 경우 10가지 프레임에 색상의 변화가 거의 없고, 간소한 움직임을 생긴 Idle107, Idle108, Idle109번, Idle110 프레임에서만 변화된 이미지 사이즈가 변화됨을 볼 수 있었다. 그렇다면 좀더 변화가 심한 대기화면 ②와 대기화면 ③의 경우를 살펴보도록 하겠다.

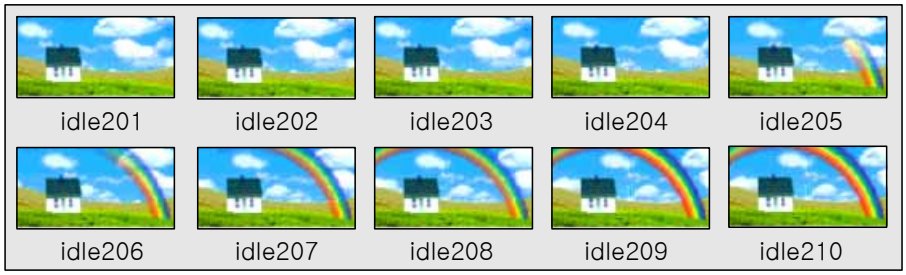

그림 5-3. 대기화면 ②의 프레임

위 그림 5-3 대기화면 ②의 경우에는 표 5-2와 함께 비교해볼 때 무지개가 점차 생겨나면서, 이미지의 모양과 색상에 변화가 점점 커짐에 따라 이미지의 데이터 증가하고 있음을 볼 수 있다. 특히 마지막 프레임에서는 압축율이 55%까지 떨어지는 점을 확인할 수 있었다.

표 5-2. 대기화면 ②의 데이터 압축율 비교표

그림 5-3의 대기화면, 101×52 크기 이미지 (바이트 단위))

파일 이름	비트맵 이미지			GIF 이미지			압축율 (%)
	파일 크기	헤더	데이터	파일 크기	헤더	데이터	
Idle201	6,486	1,078	5,408	2,696	791	1,905	65%
Idle202	6,486	1,078	5,408	2,696	791	1,905	65%
Idle203	6,486	1,078	5,408	2,696	791	1,905	65%
Idle204	6,486	1,078	5,408	2,665	791	1,874	65%
Idle205	6,486	1,078	5,408	2,938	791	2,147	60%
Idle206	6,486	1,078	5,408	3,059	791	2,268	58%
Idle207	6,486	1,078	5,408	3,077	791	2,286	58%
Idle208	6,486	1,078	5,408	3,161	791	2,380	56%
Idle209	6,486	1,078	5,408	3,203	791	2,412	55%
Idle210	6,486	1,078	5,408	3,203	791	2,412	55%

그림 5-4. 대기화면 ③의 프레임

그리고, 계절에 따라 이미지가 완전히 변환되는 위의 그림 5-4 대기화면 ③의 경우에는 압축율이 얼마나 떨어질 수 있는지에 대해서

데이터 사이즈를 표 5-3과 같이 측정해보았다.

위의 그림 5-4 대기화면 ③의 경우에는 표 5-3과 함께 비교해 볼 때 계절이 바뀌어 가면서, Idle305의 이미지에서는 봄과 여름의 두 가지의 색상을 포함할 때, 44%의 압축율로 가장 저조한 압축율을 보이는 것을 알 수 있었다. 그러나, 이와 반대로 동일한 색상이 자주 반복되는 이미지를 표현할 때는 압축율이 급격히 상승함을 마찬가지로 알수 있었다.

표 5-3. 대기화면 ③의 데이터 압축율 비교표

(그림 5-4의 대기화면, 101×52 크기 이미지 (바이트 단위))

파일 이름	비트맵 이미지			GIF 이미지			압축율 (%)
	파일 크기	헤더	데이터	파일 크기	헤더	데이터	
Idle301	6,486	1,078	5,408	2,773	791	1,982	63%
Idle302	6,486	1,078	5,408	2,773	791	1,982	63%
Idle303	6,486	1,078	5,408	2,773	791	1,982	63%
Idle304	6,486	1,078	5,408	3,478	791	2,687	50%
Idle305	6,486	1,078	5,408	3,845	791	3,054	44%
Idle306	6,486	1,078	5,408	3,242	791	2,451	55%
Idle307	6,486	1,078	5,408	3,216	791	2,425	55%
Idle308	6,486	1,078	5,408	3,332	791	2,541	53%
Idle309	6,486	1,078	5,408	3,332	791	2,541	53%
Idle310	6,486	1,078	5,408	3,116	791	2,325	57%

위의 결과들을 종합해볼 때, 현재 개발중인 단말기에서 사용하는 101×52 크기의 대기화면은 LZW 복원 알고리즘을 적용하여 최저

44%에서 65% 정도의 이미지 압축율을 보여주었다. 그렇지만, 위의 이미지들의 압축율을 평균 내었을 경우에는 약 60% 정도의 압축율을 보여주었다.

현재 개발중인 단말기는 각 사이즈 별로 약 270가지 이상의 이미지를 저장하고 있고, LZW 데이터로 변환했을 경우, 사용되는 총 이미지는 209 KB 정도의 메모리를 소요한다. 만약 이 이미지를 모두 비트맵으로 저장하게 된다면, 대략 690 KB 이상의 메모리를 이미지 데이터로만 저장해야 하는 것이다.

향후 개발 될 65000 칼라 이상의 단말기에서는 16 비트 칼라를 사용하기 때문에, 이미지 사이즈는 동일 LCD 사이즈에서 현재 256 칼라의 2배정도가 될 것이다. LZW 데이터 복원 알고리즘을 이용하는 것은 LCD 디스플레이 직전에 압축된 이미지 데이터를 원상태의 비트맵 이미지 데이터로 복구하는 것이기 때문에, 현재 이미 압축되어 있는 GIF 이미지를 이용하는 것과는 달리 압축 이미지를 만들어주는 LZW 압축 프로그램을 제작하여, 그 압축된 이미지를 위의 256 칼라 때와 똑같이 복원을 하게 된다면, 다시 압축하기전의 이미지 데이터를 출력하게 될 것이다. 그렇게 된다면 위에서 실험하였던 것과 비슷하거나 좀더 다양한 문자열 표의 생성으로 조금 더 높은 압축율을 보일 것이라고 생각된다.

휴대 전화 단말기의 경우 메모리 단가를 낮출 수 있다면, 가격 면에서 많은 이득을 볼 수 있게된다. 현재 개발중인 256 칼라 LCD를 사용하는 단말기의 경우에는 32 M(비트) 플래쉬 / 8 M(비트) SRAM을 사용하고 있으나 이후 65000 칼라 LCD를 사용하는 단말기에서는 TFT 64 M(비트) 플래쉬 / 16 M(비트) SRAM을 사용하게 된다.

그러나, 만약 위의 알고리즘으로 16 비트 칼라 데이터를 압축하였

을 경우 32 M(비트) 플래쉬 / 8 M(비트) SRAM으로 대체 할 수 있게 된다면, 비슷한 사양의 다른 단말기보다 저렴한 가격으로 생산할 수 있을 뿐만 아니라, 다음 표 5-4와 같이 매출할 경우 메모리 가격을 이득을 볼 수 있게 되는 것이다.

표 5-4. 플래쉬 메모리 가격대비

(2002년 연말 기준 업체 견적서 참조)

메모리 크기	가격	100만대	500만대	5000만대
32 M(비트) 플래쉬/ 08 M(비트) SRAM	6$	600만$	3000만$	3억$
64 M(비트) 플래쉬/ 16 M(비트) SRAM	12$	1200만$	6000만$	6억$

제 6 장 결 론

그레이 스케일 이전의 단말기는 이미지를 직접 디스플레이하더라도 이미지 데이터의 사이즈가 크지 않고, 오히려 디스플레이 속도가 빨라 아무런 문제가 없었다. 그러나, 256 칼라 이상의 LCD를 사용하는 단말기들이 개발되면서, 이미지 사이즈가 급격히 커졌을 뿐만 아니라, 유저 인터페이스에 많은 기능이 추가되면서, 그에 따라 더욱 많은 이미지들을 저장하여야만 했다. 본 논문에서는 GIF 이미지에서 사용되는 LZW 데이터 압축 알고리즘을 응용하여, 단말기에 저장되는 이미지 사이즈를 줄여보고자 하였다. 즉, 단말기에 GIF 이미지에서 실제 이미지 영역인 래스터 데이터 부분을 추출해 메모리에 저장시켜놓고, 단말기의 LCD의 UpdateDisplay( ) 부분에서 LZW 데이터를 호출해 비트맵 데이터로 복원한 후 LCD에 디스플레이 하도록 하는 알고리즘을 구현해보았다.

이와 같은 방법을 사용하였을 경우 비트맵 이미지를 그대로 사용하였을 때보다 평균 60% 정도의 압축율을 보였으며, 현재 개발 중인 휴대 전화 단말기에서는 이미지 데이터가 소요하는 메모리 사이즈를 690 KB 이상에서 209 KB 정도로 낮추어주는 효과를 보였다.

이것은 메모리 단가를 결정하는데 있어서 유용하게 쓰일 것으로 보인다. 그리고, 향후 65000 칼라 LCD를 사용하는 단말기에 이 알고리즘을 적용할 경우에도 동일한 방식으로 적용하면 되나, 압축된 이미지가 존재하지 않음으로, 이미지를 LZW 압축데이터로 바꾸어주는 프로그램을 제작을 해야한다. 이는 CDMA 단말기에만 국한되는 것이 아니라 GSM 단말기에서도 구현할 수 있을 것이다.

참 고 문 헌

[1] TIA/EIA/IS-95-A(version 0.03), Mobile Station-Base Station Compatibility Standard for Dual-Mode Wideband Spread Spectrum Cellular System.

[2] 김한주, 이광희, 손익수, 배홍균, “무선 인터넷 산업 현황 및 발전 전략,” 한국전자통신연구원 기술경영연구시리즈 00-06, pp. 14-19, 2000년 10월.

[3] 박도순, 홍창완, “A Study on The Image Compression using JPEG,” 과학기술연구논총, 제 2 집, pp. 49-51, 홍익대학교, 1994년.

[4] E. J. Delp and O. R. Mitchell, "Image compression using DCT," IEEE. Trans. Commun., Vol. COM-29, pp. 1335-1342, Dec, 1972.

[5] 김창복, “A Study on Lossless Compression of Binary Image Data,” 과학기술연구논총, 제 4 집, pp. 319-322, 홍익대학교, 1994년.

[6] Nelson, Mark R., "LZW Data Compression," Dr, Dobb's Journal, pp. 29-36, 86-87, Oct, 1989.

[7] A. N. netravali and J. D. Limb, "Picture coding : A review," Proc. IEEE. Vol. 68, pp.366-405, Mar, 1980.

[8] Graphics Interchange Format (version 87a), "A standard defining a mechanism for the storage and transmission of raster-based graphics information," CompuServe incorporated Columbus, June 15, 1987.

[9] Graphics Interchange Format (version 89a), "Programming Reference," CompuServe incorporated Columbus, July 31, 1990.

[10] Nelson, Mark R., "LZW Patent Issues," Dr. Dobb's Journal, pp. 8-12, Dec, 1989.

[11] Battilana, Michael C., "LZW Data Compression without Hasing," University of Udine Exam Project, July 9, 1987

[12] J. Ziv and A. Lempel, " A Universal Algorithm for Sequential Data Compression," IEEE Transactions on Information Theory, Vol. 23, pp.337-342, 1977.

[13] J. Ziv and A. Lempel, "Compression of Individual Sequences Via Variable-Rate Coding," IEEE Transactions on Information Theory, Vol. 24, pp. 530-536, 1978.

[14] T. A. Welch, "A Technique for High-Performance Data Compression," Computer, pp. 8-18, 1984.

[15] Wilson MacGyver Liaw, "Reading GIF Files," Dr, Dobb's Journal, pp. 64-68, Feb, 1995.

[16] TIA/EIA/IS-95-A(version 0.03). SII CONFIDENTIAL / TENTATIVE S-43200A Rev.2.0 256-COLOR, 1/80-DUTY LCD CONTROLLER/DRIVER WITH ON-CHIP RAM ESCRIPTION