선린인터넷고등학교 조강준
Unifox July 2018

PNG 파일구조의 이해와 압축 기법에 대한 연구 및 이미지 데이터를 사용하는 소스코드의 구현

목차

1 소개 . . . 3
1.1 이 프로젝트를 진행한 이유 . . . 3
1.2 제작 기간 . . . 3
1.3 앞서 설명할 내용 및 출처 . . . 3
2 PNG 파일의 장점 및 특징 . . . 3
2.1 PNG 이미지 포맷을 선택한 이유 . . . 3
2.2 PNG 이미지 포맷의 장점 . . . 4
3 PNG 이미지 파일의 구조 . . . 4
3.1 파일 시그니쳐 . . . 4
3.2 이미지 청크 . . . 5
3.2.1 청크의 기본 구조 . . . 5
3.2.1.1 Length . . . 5
3.2.1.2 Chunk type . . . 5
3.2.1.3 Chunk data . . . 5
3.2.1.4 CRC . . . 5
3.2.2 IHDR 청크 . . . 5
3.2.2.1 Width & Height . . . 6
3.2.2.2 Bit depth & Color type . . . 6
3.2.2.3 Compression method . . . 7
3.2.2.4 Filter method . . . 7
3.2.2.5 Interlace method . . . 7
3.2.3 IDAT 청크 . . . 7
3.2.4 PLTE 청크 . . . 8
3.2.5 IEND 청크 . . . 9
3.3 필터링 메소드 . . . 9
3.3.1 Adaptive filtering . . . 9
3.4 압축 메소드 . . . 11
3.4.1 Deflate / Inflate . . . 11
3.4.1.1 허프만 부호화 . . . 11
3.4.1.2 LZ77 알고리즘 . . . 11
4 ASCII Art Maker . . . 12
4.1 계획 A . . . 12
4.1.1 데이터 추출 . . . 12
4.1.2 압축 해제 및 재필터링 . . . 12
4.1.3 한계 . . . 12
4.2 계획 B . . . 12
4.2.1 알고리즘 설명 . . . 13
5. 보고서를 마무리하며 . . . 13

1 소개

1.1 이 프로젝트를 진행한 이유

이 프로젝트를 진행하게 된 이유는 주변 친구들이 C언어 프로젝트를 제작할 때 메인화면이나 아이콘을 넣기 위해 파일을 ASCII 코드로 변환하는 사이트를 보게 되었고, "이미지 파일을 PNG 파일로 바꿔보자" 라는 막연한 시도를 하게 되었다.

1.2 제작 기간

29 June : 주제 선정 및 PNG 파일 구조 연구 시작

1 July : 학업을 위한 연구 중단

14 July : IDAT chunk filtering method에 대한 연구 및 정리

15 July : Deflate / Inflate Compression method에 대한 연구 및 보고서 작성

1.3 앞서 설명할 내용 및 출처

이 보고서는 다음의 내용을 참고하여 제작되었습니다.

- https://ryanking13.github.io/2018/03/24/png-structure.html
- http://www.digital-brain.net/board/2305
- https://ko.wikipedia.org/wiki/PNG
- https://www.w3.org/TR/PNG/
- https://www.w3.org/TR/PNG-Introduction.html
- ftp://ftp.info-zip.org/pub/infozip/
- http://uzys2011.tistory.com/152

2 PNG 파일의 장점 및 특징

2.1 PNG 이미지 포맷을 선택한 이유

1999년 유니시스사에서 만든 Portable Network Graphics(이하 PNG) 이미지 포맷은 Graphics Interchange Format(이하 GIF)의 한계를 극복하기 위해 만들어졌다.
요즘에 주로 사용하는 이미지 파일 포맷은 PNG와 JPG가 있다. 손실 압축 기법을 사용하는 JPG , JPEG 이미지 포맷과 다르게, PNG 이미지 포맷은 비손실 압축 기법을 사용한다. 그렇기에 이미지가 무조건 손상되는 JPG, JPEG 파일 포맷을 사용하는 것 보다 낫다고 생각하여 PNG 이미지 파일 포맷을 선택하게 되었다.

2.2 PNG 이미지 포맷의 장점

1. 무손실 압축방식을 사용한다.

앞서 설명했듯이, 무손실 압축기법인 deflate 압축 메소드를 사용한다. 이는 zip처럼 파일이 훼손되면 안 되는 파일의 압축 메소드로 자주 사용된다. deflate는 뒤에서 더 자세히 다루어보자.

2. 압축률이 높아 파일의 크기가 작다.(GIF 기준)

대부분의 PNG 이미지 파일들은 JPG보다 이미지 파일이 크다. 하지만 이는 무손실 압축 기법에서 나온 이유기도 하다. 그리고 JPG의 압축 알고리즘은 손실 압축 알고리즘이기 때문에 JPG에서는 100% 압축으로도 무조건 데이터가 손실된다. 또한, JPG 이미지 파일의 크기보다 적은 크기를 사용할 때도 있다.

3. 풀 컬러를 지원한다.

256 RGB 색상을 지원하는 true color 색상 모드를 지원한다. 이에 대해서는 나중에 더 자세하게 다룰 내용이다.

4. 알파채널을 지원한다.

알파 채널은 투명도 채널을 일컫는다.

5. 인코딩 & 디코딩 속도가 빠르다

빠른 압축 알고리즘을 사용하여 인코딩&디코딩 속도가 빠른 편이다.

6. 오픈 라이선스이다.

GIF와 다르게 오픈 라이선스 이므로 저작권 없이 사용할 수 있으며, 이는 PNG 이미지 포맷의 접근성을 높이는 요소이기도 하다.

3 PNG 이미지 파일의 구조

PNG 이미지 파일은 파일 시그니쳐와 여러 개의 청크로 이루어진다.

3.1 파일 시그니쳐

모든 파일의 앞에는 8byte의 파일 시그니쳐라는 것이 붙게 된다. 그로써 앞으로 어떻게 파일을 해석해야 하는가에 대해 알 수 있게 된다. PNG 파일의 시그니쳐는 “89 50 4E 47 0D 0A 1A 0A” 이다. 그 중 “89 50 4E” 는 ASCII로 PNG 이다.

3.2 이미지 청크

3.2.1 청크의 기본 구조

PNG 이미지 파일을 해석할 때 한 종류의 공통된 데이터를 가지고 있는 데이터의 집합을 '청크' 라고 한다. 여기에는 대표적으로 IHDR, IDAT, PLTE, IEND가 있다. 중요 청크는 앞 글자가 대문자이며, 보조청크는 앞 글자가 소문자이다. 청크는 일정 부분이 동일한 크기로 이루어져 있다. 청크의 구조는 다음과 같다.

```
Chunk(12 + length byte) {
  Length(4 byte),
  Chunk Type (4 byte),
  Chunk Data (length byte),
  CRC (4 byte)
}
```

3.2.1.1 Length

Chunk Data의 크기를 지정해준다. Length를 표현하는 데 4 byte를 사용한다.

3.2.1.2 Chunk type

IHDR, IDAT 같은 이미지 파일의 타입을 지정해준다. 4byte를 차지하며, ASCII 코드로 작성된다.

3.2.1.3 Chunk data

실제로 데이터가 들어가는 부분이며, Length byte를 차지한다. 즉 가변길이이다. 안에 들어갈 내용은 청크 타입에 따라 다르다.

3.2.1.4 CRC

CRC는 길이를 제외한 청크 타입과 데이터로 생성된 network byte order CRC32이다. 크기는 4byte이며, 본 보고서에서는 CRC를 자세하게 다루지 않도록 하겠다.

3.2.2 IHDR 청크

Image header (이하 IHDR) 청크는 파일 시그니쳐 바로 뒤에 붙는다. 즉, 실질적 데이터의 가장 첫 번째 이다. IHDR 청크에는 이미지 파일의 기본정보들인 가로, 세로, 비트 깊이, 색 타입, 압축 메소드, 필터링 메소드, 인터레이스 메소드에 대한 정보가 담겨있다. IHDR 청크는 다음과 같이 이루어져있다.

IHDR{

Length : 00 00 00 0D (4 byte),

Chunk Type : IHDR(4 byte),

Chunk Data (13 byte : 고정 길이),

{

Width (4 byte),

Height (4 byte),

Bit depth (1 byte),

Color Type (1 byte),

Compression method (1 byte),

Filter method (1 byte),

Interlace method (1 byte),

}

CRC (4 byte)

}

3.2.2.1 Width & Height

말 그대로 이미지의 폭과 넓이에 대해 기술되어있다. 각각 4byte를 차지한다.

3.2.2.2 Bit depth & Color type

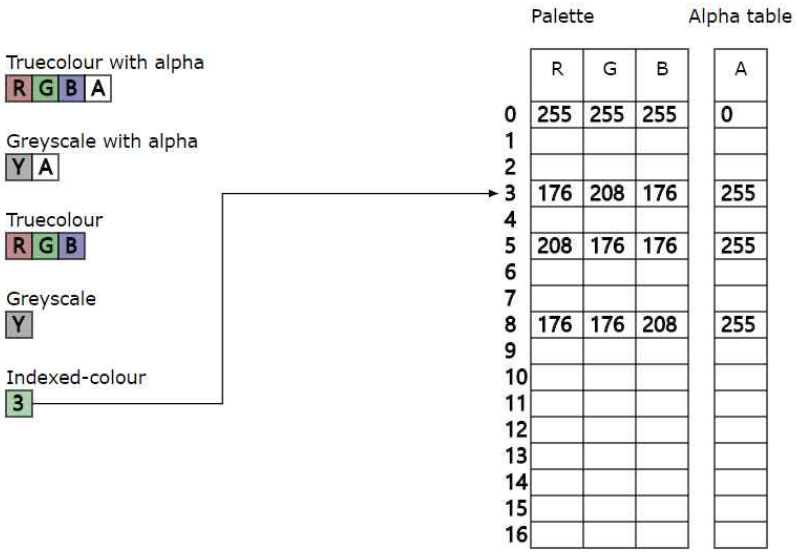
Color type은 PNG 이미지의 색상을 어떻게 구성할 것인지를 정하고,

Bit depth 하나의 채널(channel)이 몇 비트로 구성될 지를 정한다.

Color type과 Color type에 대한 bit depth는 다음과 같이 정의된다.

PNG image type	Color Type	Allowed bit depths	interpretation
Grayscale	0	1, 2, 4, 8, 16	Grayscale Data
Truecolor	2	8, 16	RGB Data
Indexed-color	3	1, 2, 4, 8	Palette Data
Grayscale with alpha	4	8, 16	Grayscale + Alpha Data
Truecolor with alpha	6	8, 16	RGB + Alpha Data

<표 1> 색상 유형과 그에 따른 비트 심도



<사진 1> 색상 유형 픽셀 구조

3.2.2.3 Compression method

현재까지 PNG에서 표준으로 정의된 압축 방식은 0: [DEFLATE]한 가지이다.

3.2.2.4 Filter method

현재까지 PNG에서 표준으로 정의된 필터링 방식은 0: [Adaptive Filtering] 한 가지다.

3.2.2.5 Interlace method

인터레이스는 웹 페이지 등에 이미지를 표시할 때 이미지 로딩이 완료되기 전 먼저 해상도가 낮은 이미지를 보여주기 위하여 사용된다. 현재까지 PNG에서 표준으로 정의된 인터레이스 방식은, 0 (No interlace), 1 (Adam7 interlace) 두 가지다.

3.2.3 IDAT 청크

Image data (이하 IDAT) 청크는 이미지의 실질적인 데이터를 가지고 있다. IDAT 청크는 압축되어있기 때문에 읽을 때 decoding 과정이 필요하다. (IDAT 청크 데이터의 압축 해제 변환과, 압축 변환에 대한 내용은 뒤에서 자세히 다루도록 한다.)
한 PNG 파일은 여러 개의 IDAT 청크를 가질 수 있는데, 이는 데이터를 *스트리밍 방식으로 전송하기 위한 것이다. IDAT 방식을 사용하면 PNG를 으로 전달할 수 있게 한다(ㅋ 인터레이스). 또한 하나의 IDAT 청크가 이미지의 특정 부분을 나타내는 것은 아니기 때문에 모든 IDAT 청크가 있어야만 이미지 디코딩을 정상적으로 할 수 있다.

IDAT 청크는 다음과 같은 구조로 이루어져있다.

```
IDAT{
  Length (4 byte),
  Chunk Type : IDAT(4 byte),
  Chunk Data (Length byte),
  {
        Filtered, Compressed Data
        .
        .
  }
  CRC (4 byte)
}
```

*스트리밍 방식: 인터넷에서 데이터를 실시간 전송, 구현할 수 있게 하는 기술로 데이터를 연속적으로 전송하여 실시간으로 재생하는 일.

3.2.4 PLTE 청크

Palette (이하 PLTE) 청크는 팔레트, 즉 색 공간을 표시한다. color type 3에서는 색을 표시하기 위해 항상 PLTE 청크가 필요하고, color type 2또는 6에서는 선택적으로 필요하다. PLTE 청크는 bit depth에 관계없이 한 샘플(색상 채널 ex. R, G, B, A) 당 8 비트를 사용한다. 팔레트 항목 수는 bit depth로 표현할 수 있는 범위를 초과하지 않는다. PLTE 청크는 다음과 같다.

```
PLTE{
  Length (4 byte),
  Chunk Type : IDAT(4 byte),
  Chunk Data (Length byte),
  {
        RGB palette[0] (3 byte),
        RGB palette[1] (3 byte),
        .
        .
  }
  CRC (4 byte)
}
```

3.2.5 IEND 청크

IEND 청크는 이미지의 맨 뒤에 위치하는 청크로, 이미지 파일의 끝을 표시하는 청크이다. 아무 데이터가 없기 때문에 길이가 0이다. IEND 청크의 구조는 다음과 같다.

IEND{
 Length : 00 00 00 00 (4 byte),
 Chunk Type : IEND (4 byte),
 Chunk Data (0 byte),
 CRC (4 byte)
}

3.3 필터링 메소드

3.3.1 Adaptive Filtering

앞서 말했듯 현재까지 PNG에서 표준으로 정의된 필터링 방식(0: [Adaptive Filtering])은 이미지의 각 행(*scanline)에 대하여 5가지 필터 타입 중 한 가지를 정하여 바이트 단위로 적용한다. 필터링 할 바이트를 x라고 할 때, a,b,c를 다음과 같은 상대적 위치로 정의한다. *scanline: 한 줄을 기준으로 인코딩을 진행하는 방식.

Name	Position
x	필터링 할 바이트
a	x의 바로 왼쪽 바이트
b	x의 바로 위 바이트
c	b의 바로 왼쪽 바이트

<표 2> x를 기준으로 한 a,b,c의 상대적 위치

시각적으로 표현하면 아래와 같이 표현 할 수 있다.

---

d: 해당사항 없는 바이트

d d d d
d c b d
d a x d
d d d d

그리고 위와 같이 정의되는 값을 바탕으로 아래의 5가지 필터 타입을 적용한다.

Type	Name	Filter Function	Reconstruction Function
0	None	Filt(x) = Orig(x)	Recon(x) = Filt(x)
1	Sub	Filt(x) = Orig(x) - Orig(a)	Recon(x) = Filt(x) + Recon(a)
2	Up	Filt(x) = Orig(x) - Orig(b)	Recon(x) = Filt(x) + Recon(b)
3	Average	Filt(x) = Orig(x) - floor((Orig(a) + Orig(b)) / 2)	Recon(x) = Filt(x) + floor((Recon(a) + Recon(b)) / 2)
4	Paeth	Filt(x) = Orig(x) - PaethPredictor(Orig(a), Orig(b), Orig(c))	Recon(x) = Filt(x) + PaethPredictor(Recon(a), Recon(b), Recon(c))

<표 3> Aaptive 필터의 5가지 필터 타입

Filt(x): 필터가 적용된 후의 x의 값

Orig(x): 바이트 x의 원본(필터링 되지 않은 값)

Recon(x): 재구성된 기능이 적용된 후의 x의 값

필터링 알고리즘은 Bit depth 또는 Color type에 관계없이 **픽셀이 아닌 바이트** 에 적용되는 것에 주의한다. 아래는 PaethPredictor(a,b,c)의 어셈블리 코드이다.

```
---------------------------------------------
function PaethPredictor (a, b, c)
    begin
        ; a = left, b = above, c = upper left
        p := a + b - c          ; initial estimate
        pa := abs(p - a)        ; distances to a, b, c
        pb := abs(p - b)
        pc := abs(p - c)
        ; return nearest of a,b,c,
        ; breaking ties in order a,b,c.
        if pa <= pb AND pa <= pc then return a
        else if pb <= pc then return b
        else return c
    end
---------------------------------------------
```

3.4 압축 메소드

압축 메소드도 필터링 메소드처럼 PNG에서 표준으로 정의된 압축 방식은 Deflate Compression으로 한 가지이다.

3.4.1 Deflate / Inflate

압축 해제 기법은 inflate라고 하며, deflate/inflate 압축 알고리즘은 LZ77의 파생 알고리즘이다. 압축한 데이터는 다음과 같은 구조를 갖는 "zlib" 형식으로 저장된다.

{

Compression method/flags code (1 byte)

Additional flags/check bits (1 byte)

Compressed data blocks (n byte)

Check value (4 byte)

}

zlib 데이터 스트림 내의 압축 된 데이터는 일련의 블록으로 저장 되고, 각 블록은 LZ77압축 데이터를 허프만 코드로 인코딩한 데이터로 이루어진다. LZ77 및 허프만 부호화 알고리즘의 인코딩 방식보다는 디코딩 방식을 위주로 설명할 것이다. 인코딩 방식의 자세한 내용은 RFC-1951에 기술되어있다.

3.4.1.1 허프만 부호화 알고리즘

허프만 부호화 알고리즘을 간략히 설명하자면, 문자열에서 사용된 문자열의 빈도를 기준으로 허프만 트리를 만들어 빈도수가 높은 문자부터 길이가 짧은 논리 코드를 할당하여 압축하는 방식이다. 압축을 해제할 때는 문자와 코드를 알려주는 허프만 테이블을 참조하여 디코딩한다.

3.4.1.2 LZ77 알고리즘

말로만 간단히 설명하자면 이미 나왔던 **문자열**이 나오면 그 위치만 표현해주는 알고리즘이다. 문자열 압축 알고리즘이기 때문에 압축률이 높은것이 특징이다.
디코딩하는 방법은 그냥 처음부터 순서대로 읽어나가기만 하면 된다. 자세한 내용은 출처를 참고해보자.

4 ASCII Art Maker

4.1 계획 A

드디어 방대한 양의 서론(?)을 끝마치고 드디어 PNG를 ASCII로 바꾸는 프로그램에 대한 설명이다. 구조는 간단(?)하다.

1. 이미지 파일을 이진 파일(binary file)로 불러온다.
2. 파일을 청크 단위로 쪼개 읽은 뒤 IDAT 청크의 데이터를 하나의 배열로 모은다.
3. 모은 데이터를 비트 단위로 쪼개어 원본 데이터 더미로 가공한다.
4. 원본 데이터를 다시 RGB 단위로 쪼갠다.
5. 드디어 RGB 데이터를 가지고 아스키 아트를 만든다.

4.1.2 데이터 추출

휴리스틱적으로 데이터를 추출해보자면, 실질적 데이터가 들어있는 IDAT 파일의 (PLTE 청크는 다루지 않도록 하겠다.) 모든 정보를 한군데 모으면 압축되어있는 데이터를 추출할 수 있게 된다.

4.1.2 압축 해제 및 재 필터링

앞서 RGB 데이터는 필터링과 Deflate라는 압축 알고리즘에 의해 가공된다고 말했었다. 그렇다면 우리는 RGB값을 얻으려면, 원본 데이터에 Inflate(압축: deflate, 압축 해제: inflate) 알고리즘을 적용시키고, 재 필터링을 해주면 나올 것이다.

4.1.3 한계

내가 아무리 구글링을 해봐도 infate할 때 허프만 트리를 어떻게 참조하는지 잘 안 나오고, 나온다 하더라도 설명을 잘 알아들을 수 없는 관계로 이 방법으로는 포기하게 되었다.

4.2 계획 B

c++로는 부족할 것 같아서 python의 image 모듈을 참고하여 소스를 작성하였다. 이번에는 더 간단하다.

1. 이미지를 이진 파일로 연다.
2. 이미지 파일의 픽셀 값을 받아온다.
3. 이미지가 가져온 픽셀 값을 바탕으로 아스키 코드를 작성한다.

4.2.1 알고리즘 설명

알고리즘도 매우 간단하다. 일단 모든 RGB 데이터를 가져온다. 픽셀 데이터의 디코딩은 모듈이 알아서 다 처리해준다. 그리고 색상을 표현할 수는 없으니, RGB 데이터의 평균을 아스키 코드로 표현하여 텍스트 파일에 저장하였다. 이보다 간단할 수 없다.

5 보고서를 마무리하며

일단 이 보고서는 형식상 첫 번째 보고서이다. ASCII Art의 취지와는 약간 벗어나고, 마지막에는 알고리즘을 제대로 이용하여 만든 것은 아니지만, 나는 PNG 파일을 조사하며 구조, 압축 알고리즘 등등 많은 것을 알게 되었으며, 이는 꼭 부정적으로만 바라보지는 않겠다. 만약 이 프로젝트를 진행할 시간이 더 주어진다면, 그때는 정말 디코딩 방식으로 라이브러리나 모듈 없이 ASCII Art를 구현해보고싶다.