PNG にひそむ ZIP の技術

ゆずねり　著

2026-11-21 版　　NeIn 発行

はじめに

スクリーンショットを保存するとき、ロゴを書き出すとき、私たちは何気なく PNG を選びます。写真は JPEG、それ以外は PNG。では、PNG はなぜ 1 ピクセルも変えずにファイルを圧縮できるのでしょうか。

PNG は 1995 年、GIF の特許問題への抗議から、わずか 2 か月の議論で設計されました。その圧縮の中身を開けてみると、ZIP や gzip と同じ Deflate が入っています。PNG は画像の性質を利用してデータを偏らせ、汎用の圧縮方式に渡します。

本書では、PNG の誕生の経緯から、チャンクと呼ばれるファイル構造、フィルターと Deflate による圧縮の仕組み、そして 2025 年の PNG 仕様第 3 版で正式になったアニメーション PNG までを解説します。

PNG にひそむ ZIP の技術を、一緒に解き明かしていきましょう。読み終えるころには、見慣れた PNG ファイルが少し違って見えてくるはずです。

本書サポートサイトのご案内

本書のサポートサイトは次の URL です。

`https://neln.net/b/nt06/`

免責事項

本書に記載された内容は、情報の提供のみを目的としています。したがって、本書を用いた開発、製作、運用は、必ずご自身の責任と判断によって行ってください。これらの情報による開発、製作、運用の結果について、著者はいかなる責任も負いません。

目次

はじめに **2**

第 1 章 PNG 誕生 **6**
1.1 PNG の誕生 6
1.2 ZIP との縁 7
1.2.1 LZW 特許と gzip 7
1.2.2 Info-ZIP と PNG 7
1.2.3 zlib の誕生 8
1.3 PNG の規格 8
1.3.1 PNG 1.0 8
1.3.2 PNG 1.1 と 1.2 8
1.3.3 第 2 版 9
1.3.4 第 3 版 9

第 2 章 PNG の内部構造 **10**
2.1 PNG のシグネチャー 10
2.2 PNG のチャンク 11
2.2.1 チャンクタイプの大文字と小文字 12
2.2.2 必須チャンク 13
2.2.3 補助チャンク 15

第 3 章 データを圧縮する仕組み **21**
3.1 デジタル画像の圧縮 21
3.2 データの冗長性 22
3.3 冗長性を省く三つの手法 22
3.3.1 ランレングス符号化（RLE） 22
3.3.2 辞書式圧縮 23
3.3.3 ハフマン符号化 23

3.4 PNG における圧縮の組み合わせ 24

第 4 章 PNG 圧縮アルゴリズム 25

4.1 PNG 圧縮の考え方 25
4.1.1 予測して差分だけを記録 25
4.2 圧縮処理の流れ 26
4.3 インターレース 26
4.4 フィルタリング 28
4.4.1 参照する近傍 28
4.4.2 5 種類のフィルター 28
4.4.3 差分から元の値を復元 29
4.4.4 フィルターごとの予測の違い 30
4.4.5 フィルターの選び方 31
4.4.6 予測によって生まれる偏り 31
4.4.7 圧縮サイズへの効果 32
4.4.8 上の行の参照を LZ77 に任せられない理由 32
4.5 Deflate 圧縮 33
4.5.1 zlib 形式 33
4.5.2 zlib、gzip、ZIP の違い 34
4.5.3 ブロック構造 35
4.5.4 LZ77 による分解 36
4.5.5 長さと距離の表現 36
4.5.6 ハフマン符号化 38
4.5.7 マッチとリテラルの損得 41
4.6 全体の計算過程 41
4.6.1 写真に近い例 41
4.6.2 図形やイラストに近い例 44
4.6.3 フィルターを使わない場合 49
4.6.4 カラータイプによるサイズの違い 50
4.7 エンコーダーパラメーター 51
4.7.1 フィルターの候補 51
4.7.2 zlib の圧縮レベル 52
4.7.3 zlib の戦略 52
4.7.4 レベルと戦略によるサイズの違い 53
4.7.5 LZ77 のウィンドウサイズ 54

第 5 章 アニメーション PNG 56

5.1 アニメーション PNG とは 56
5.2 三つのチャンク 57
5.2.1 acTL（アニメーション制御） 57
5.2.2 fcTL（フレーム制御） 57
5.2.3 fdAT（フレームデータ） 58
5.2.4 1 フレーム目と IDAT 59
5.3 フレームの合成規則 59
5.3.1 blend_op（合成方法） 59
5.3.2 dispose_op（後始末） 60
5.4 GIF や WebP のアニメーションとの違い 61

あとがき **63**

第 1 章

PNG 誕生

PNG は特許問題への抗議をきっかけに生まれた画像フォーマットです。

本章では 1995 年に起きた騒動から、PNG の誕生の経緯をたどります。圧縮方式に ZIP や gzip と同じ Deflate が選ばれた理由とその選択を支えた人々の取り組み。そして W3C 勧告から ISO 標準化を経て、2025 年の第 3 版に至るまでの規格の歩みを振り返ります。

1.1 PNG の誕生

1990 年代半ば、Web で使われるカラー画像フォーマットの主役は GIF でした。GIF の圧縮には、Unisys 社と IBM 社が特許を持っている LZW（Lempel-Ziv-Welch）アルゴリズムが使われていました。

1994 年 12 月 29 日、GIF の開発元である CompuServe 社は Unisys 社との特許ライセンス契約に基づき、それまで自由に使えていた GIF 対応ソフトウェアの開発者に、ライセンス料の支払いを求めました[*1]。

この発表に、ネットニュースのコミュニティーは猛反発しました。やがて、特許に縛られず、256 色までしか扱えない GIF より高機能なフォーマットを開発しようという動きが広がります。

1995 年 1 月 4 日、Thomas Boutell 氏が新フォーマットの最初の草案をネットニュースに投稿しました。これに世界中の開発者が反応し、ネットニュースとメーリングリスト上の議論だけで設計が進んでいきました。本書で解説するチャンク構造、フィルター、Deflate といった基本の仕組みは、投稿から数週間の議論で提案されたものです。

[*1] 発端となった Unisys 社の LZW 特許は、米国では 2003 年、日本では 2004 年に失効しています。

最初の投稿からわずか 2 カ月後の 1995 年 3 月 7 日、第 9 草案で仕様がほぼ確定しました。こうして生まれたのが Portable Network Graphics、PNG です[*2]。

1.2 ZIP との縁

PNG の圧縮には、ZIP や gzip と同じ Deflate という方式が使われています。当時の提案者や設計者が挙げた理由は、主に三つです。LZW のような特許の問題を避けられると見込まれたこと、GIF の LZW より高い圧縮率が得られること、無償で使える実装がすでにあったことです。その判断の材料となった特許の調査や Deflate の実装を担った人々は、PNG の設計議論にも加わっていました。

1.2.1 LZW 特許と gzip

GIF の騒動に先立って、UNIX の世界でも LZW 特許が問題になっていました。UNIX の標準的な圧縮コマンド compress は LZW を使っており、同じ特許の対象だったのです。

1992 年 10 月、Jean-loup Gailly 氏と Mark Adler 氏は compress を置き換えるため、gzip を公開しました。gzip の圧縮方式として採用されたのが、Phil Katz 氏が自身のアーカイバー「PKZIP」のために設計した Deflate です。PKZIP は、ZIP 形式を生み出したソフトウェアです。Deflate は、特許に抵触しない方法で実装できると考えられていました[*3]。

1.2.2 Info-ZIP と PNG

Deflate を広めたもう一つの存在が、PKZIP 互換の無償ツールを開発していた Info-ZIP グループです。Gailly 氏と Adler 氏もこのグループのメンバーでした。

最初の草案が投稿された週には、Independent JPEG Group を率いる Tom Lane 氏や Info-ZIP グループをはじめ、多くの参加者が Deflate を提案しました。Lane 氏は、gzip が特許に抵触しないことの裏づけを開発者が重ねている点と、LZW より圧縮率が高い点を挙げます。Info-ZIP の Greg Roelofs 氏は、GIF の LZW を Deflate に置き換えると圧縮率が約 10% 向上するという試験結果を投稿しました。Boutell 氏も第 3 草案で、Info-ZIP の zip/unzip に商用利用も認める、寛容なライセンスの実装があることを挙げています。

Info-ZIP のメンバーの貢献は圧縮方式にとどまりません。Roelofs 氏はデータの

[*2] 非公式には「PNG's Not GIF（PNG は GIF ではない）」とも呼ばれます。

[*3] Deflate を規定する RFC 1951 は、この形式を「特許に抵触しない方法で容易に実装できる」ものとしています。`https://www.rfc-editor.org/rfc/rfc1951`

第 2 章

PNG の内部構造

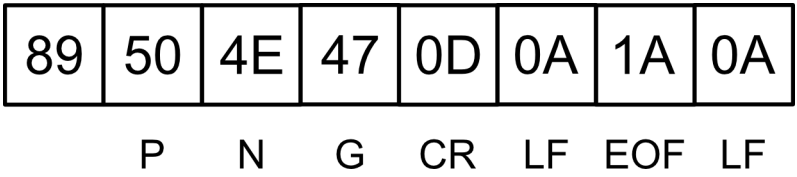
PNG ファイルはシグネチャーと、それに続く複数のチャンクで構成されています。

本章では PNG ファイルをバイト列として読み解きます。先頭 8 バイトのシグネチャーに込められた工夫、データを格納するチャンクの構造を順に見ます。

2.1 PNG のシグネチャー

PNG ファイルは、PNG 形式であることを識別する 8 バイトのシグネチャーで始まります。

シグネチャーの内容は次のとおりです。



▲図 2.1　PNG のシグネチャー

この 8 バイトは 1990 年代のファイル転送で起こりがちだった事故を検出できるように設計されています。ASCII が 7 ビットで足りたため 8 ビット目を保証しなかった通信経路で最上位ビットが落ちる、テキストファイルと誤認されて改行コードが変換される、といった事故です。各バイトの役割は次のとおりです。

▼表 2.1　シグネチャーの各バイトの役割

バイト目	値	意味	役割
1	0x89	ASCII の範囲外の値	最上位ビットの欠落や、テキストとの誤認に備える
2〜4	0x50 0x4E 0x47	ASCII の「PNG」	人間が見ても識別できる
5〜6	0x0D 0x0A	DOS 形式の改行（CR + LF）	LF に変換されると CR が失われ、検出できる
7	0x1A	EOF（DOS で表示を終了する制御文字）	バイナリーが画面に流れるのを防ぐ
8	0x0A	UNIX 形式の改行（LF）	CRLF に変換されると CR が挿入され、検出できる

CR（キャリッジリターン、\r）と LF（ラインフィード、\n）は改行の制御文字で、EOF は End of File の略です。

先頭の 0x89 は、2 進数では 10001001 です。最上位ビット（左端の 1）が 7 ビット経路で失われると 0x09 に変わるため、シグネチャーの不一致からデータの破損に気づけます。また、ASCII の範囲（0x00 から 0x7F）の外の値を置くことで、テキストファイルとの誤認も防ぎやすくしています。

2.2　PNG のチャンク

シグネチャー以降のデータは、チャンクという単位で構成されます。各チャンクは長さ（Length）、チャンクタイプ（Chunk Type）、データ（Data）、CRC（Cyclic Redundancy Check）の四つの部分で構成されます。

▼表 2.2　PNG チャンクの構成

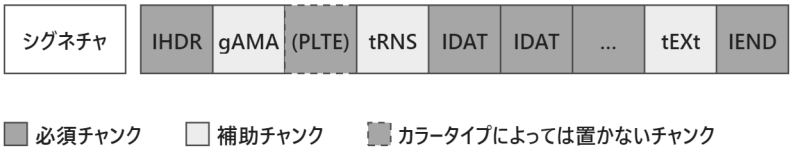
部分	バイト数	説明
長さ（Length）	4	データ部のバイト数を示す 4 バイトの整数
チャンクタイプ（Chunk Type）	4	チャンクの種類を示す 4 バイトの ASCII 文字
データ（Data）	可変	チャンクの実際のデータ。長さ（Length）でバイト数が決まる
CRC（Cyclic Redundancy Check）	4	チャンクタイプとデータ部に対する CRC-32 の検査値

長さのような複数バイトの整数は、上位のバイトから順に並べます（ビッグエン

ディアン）。この並びは画像ヘッダーの幅や高さなど、PNG の仕様が定めるすべての整数フィールドに共通です。

CRC はチャンクタイプとデータ部から計算した検査値です。デコーダーが同じ計算をして比較すれば、チャンク単位でデータの破損を検出できます。

チャンクには、画像の表示に欠かせない必須チャンクと、なくても表示できる補助チャンクがあります。シグネチャーの後に必須チャンクと補助チャンクを次の図のように並べ、IEND チャンクでファイルを終えます。



▲図 2.2　PNG ファイルのチャンクの並び

2.2.1　チャンクタイプの大文字と小文字

チャンクタイプを見ると、tRNS や gAMA のように、大文字と小文字が混在しています。これは、4 文字それぞれの大文字と小文字の違いがチャンクの性質を表すビットフラグになっているためです。

- 1 文字目：大文字なら必須（Critical）、小文字なら補助（Ancillary）チャンク
- 2 文字目：大文字なら仕様で定義されたチャンク、小文字ならアプリケーションが独自に定義するプライベートチャンク
- 3 文字目：将来のための予約。現在は常に大文字
- 4 文字目：大文字なら条件付きで、小文字なら常にコピーしてよいチャンク

画像を取り出そうとするデコーダーは未知のチャンクに出会うと、1 文字目が小文字（補助）なら無視できます。大文字（必須）なら解釈できない情報が画像に含まれることを、ユーザーに示さなければなりません。中身を解釈できないチャンクであっても、先頭に長さがあるので次のチャンクの位置まで読み飛ばせます。

編集ソフトは解釈できないチャンクに出会うと、4 文字目でファイルに残してよいかを判断します。小文字なら編集の内容を問わずコピーしてかまいません。大文字のチャンクは画像データに依存する情報を持つことがあります。必須チャンクの追加、変更、削除、並べ替えのいずれかをしたときはコピーできません。

2.2.2 必須チャンク

必須チャンクは、より正確には、デコーダーが解釈できなければ画像を正しく表示できないチャンクです。IHDR、IDAT、IEND はすべての PNG ファイルに含まれますが、PLTE はカラータイプによって要否が変わります。

シグネチャーに続く必須チャンクは、次の順序で並べます。

▼表 2.3　必須チャンクの順序規則

チャンク	個数	位置
IHDR	1	シグネチャーの直後
PLTE	0 または 1	最初の IDAT より前
IDAT	1 以上	すべて連続して置く
IEND	1	末尾

この順序により、デコーダーが画像データに到達する前に、表示に必要な情報がそろいます。

IHDR（画像ヘッダー）チャンク

IHDR チャンクはシグネチャーの次に必ず配置するチャンクです。画像の基本情報を格納します。

デジタル画像はピクセル（画素）と呼ばれる、小さな点の集まりで構成されています。各ピクセルは色を数値で持ち、カラー画像では赤、緑、青（RGB）の光の強さを組み合わせることでさまざまな色を表現しています。画像の幅と高さは、このピクセルの数で表します。

▼表 2.4　IHDR のフィールド

フィールド	サイズ	内容
Width	4 バイト	画像の幅（ピクセル）
Height	4 バイト	画像の高さ（ピクセル）
Bit depth	1 バイト	ビット深度（1、2、4、8、16）
Colour type	1 バイト	カラータイプ（0、2、3、4、6）
Compression method	1 バイト	圧縮方式。0（Deflate）のみ定義
Filter method	1 バイト	フィルター方式。0 のみ定義
Interlace method	1 バイト	0（なし）または 1（Adam7）

圧縮方式とフィルター方式のフィールドは将来の拡張のために用意されています

第 3 章

データを圧縮する仕組み

PNG は画質をまったく劣化させずにファイルサイズを削減する、可逆圧縮の画像フォーマットです。

すべてのピクセルを完全に保存しているのに、なぜデータ量を減らせるのでしょうか。その鍵はデータの中に潜む繰り返しと偏りにあります。

本章では PNG の圧縮の土台として、デジタル画像のデータ量と、情報を失わずにデータ圧縮する基本を解説します。

3.1　デジタル画像の圧縮

デジタル画像の各ピクセルは色を数値で持ちます。

たとえば、1 ピクセルの色を表すのに赤、緑、青それぞれ 8 ビット（0〜255 の整数値）を使う場合、1 ピクセルあたり 24 ビット（3 バイト）のデータが必要です。1000 × 1000 ピクセルの画像なら 3,000,000 バイトで、ピクセルが増えるほど必要なデータ量も増え、保存や転送の負担になります。

このデータを圧縮するには、二つの考え方があります。

可逆圧縮（ロスレス圧縮）は、元のデータを完全に復元できる圧縮手法です。PNG はこの可逆圧縮を採用しています。

非可逆圧縮（ロッシー圧縮）は、元のデータから一部の情報を捨てることで圧縮する手法です。復元しても元のデータにはならない代わりに、ファイルサイズを可逆圧縮より大幅に小さくできます。JPEG はこの非可逆圧縮を採用しており、写真などで広く使われています。

JPEG は人間の視覚特性を利用して、見えにくい色や細かな模様を捨てることでデータを圧縮します。一方、PNG は何も捨てず、表現方法を工夫することでデータを小さくします。

3.2 データの冗長性

冗長性とは、繰り返しや出現頻度の偏りのように、データの中にある省ける無駄を指します。たとえば、「すももももももものうち」という文には、「も」が何度も繰り返されています。このような繰り返しを「もが 8 個」のようにまとめて表現すれば、同じ情報をより少ない文字で伝えられます。

同じことが画像データにも当てはまります。特にロゴやスクリーンショットでは、同じ色が広い範囲に続く領域や、同じパターンの繰り返しが多く見られます。可逆圧縮ではこの冗長性を見つけて、より短い表現に置き換えることでデータを圧縮します。

逆に、データに冗長性がほとんどない場合、可逆圧縮ではファイルサイズをあまり圧縮できません。

3.3 冗長性を省く三つの手法

冗長性を省く代表的な手法に、ランレングス符号化、辞書式圧縮、ハフマン符号化の三つがあります。

3.3.1 ランレングス符号化（RLE）

もっとも基本的な圧縮手法の一つが、ランレングス符号化（Run-Length Encoding、RLE）です。

同じデータが連続する部分（ラン）を、データと個数の形に置き換えて表現します。

前述した「すももももももものうち」は、「す 1、も 8、の 1、う 1、ち 1」という文字と個数の 5 組で表現できます。文字も個数も一つの記号と数え、記号をビットでどう表すかは問わないことにすると、記号の数は元の 12 文字より少ない 10 個になります。

ランレングス符号化はシンプルで高速なため、FAX や古い画像フォーマットなどでも広く使われてきました。

ただし、同じデータが連続している必要があります。「ABCABCABC」のようなデータでは 1 文字も縮まず、記録方法によってはかえってデータが増えます。写真のように、ノイズを含んで細かく値が変わるデータも苦手です。

第 4 章

PNG 圧縮アルゴリズム

PNG が 1 ピクセルも変えずにファイルサイズを圧縮できるのは、画像の性質を利用してデータの偏りを作り出しているからです。

本章では、どうやってデータを偏らせているのか、Deflate はどうやってデータを圧縮しているのかを順に見ます。後半では 8 × 8 ピクセルの画像 2 枚で圧縮の全過程を追い、エンコーダーのパラメーターがサイズに与える影響を見ます。

4.1 PNG 圧縮の考え方

PNG は特にロゴやイラスト、スクリーンショットのような、色の変化が少ない画像に向いています。同じ色が広い範囲に続く部分は、そのままでも繰り返しとして圧縮できます。色がなだらかに変わる部分も、隣のピクセルから予測した差分だけを記録すれば 0 付近の値が並び、圧縮しやすくなります。

4.1.1 予測して差分だけを記録

画像では、隣接するピクセルの色は似ていることが多くなります。青空のある画像では隣り合うピクセルがほぼ同じ青ですし、グラデーションでは値が一定の割合で変化します。

この性質を利用すると、各ピクセル値は「隣のピクセルから予測した値との差分」として表現できます。予測が正確であるほど差分は 0 に近くなり、0 付近に偏りが生まれます。

0 付近に偏ったデータでは、同じ値の繰り返しを LZ77 が参照に置き換え、出現頻度の高い値をハフマン符号化が短い符号に変換します。この二つの組み合わせが、Deflate 圧縮です。

4.2 圧縮処理の流れ

PNG では次のような順番で圧縮します。復号は逆順で行います。

1. インターレース（使う場合のみ）
2. フィルタリング
3. LZ77
4. ハフマン符号化

インターレースは、段階的な表示のために画像を七つのパスに分割する、省略可能な前処理です。フィルタリングでは、各バイトの値をその周囲のバイトから予測し、実際の値との差分だけを記録します。

この四つのうち、3 と 4 が Deflate 圧縮の処理です。Deflate の出力を zlib 形式で包んだものが、IDAT チャンクに格納されます。

4.3 インターレース

PNG には、粗い状態から段階的に表示するインターレース機能があります。

通信回線が遅かった時代、大きな画像は上から少しずつ表示されていました。インターレースを使うと、最初に低解像度の画像が表示され、徐々に詳細が追加されていくため、ユーザーは待ち時間を感じにくくなります。

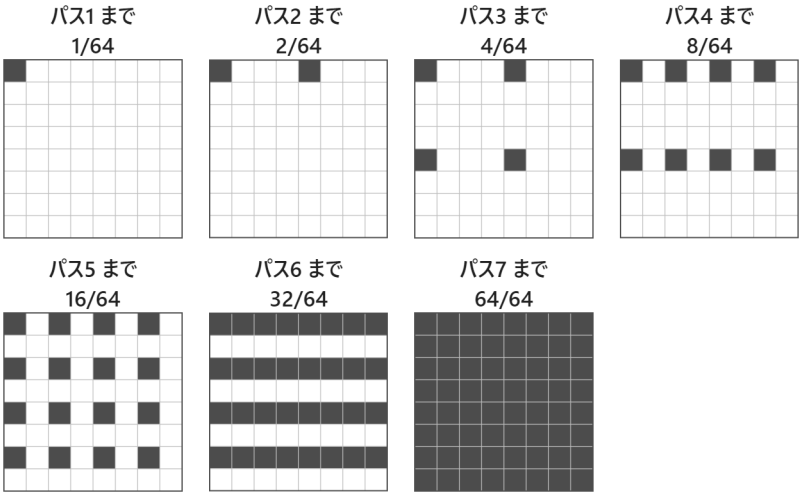
PNG のインターレースは Adam7 と呼ばれます[*1]。

画像を 8×8 のブロックに分割して、7 回にわたって順番に表示します。各パスは独立した小さな画像として扱われ、パスごとにフィルタリングされてから、まとめて Deflate で圧縮されます。

[*1] 考案者の Adam Costello 氏にちなみます。

1	6	4	6	2	6	4	6
7	7	7	7	7	7	7	7
5	6	5	6	5	6	5	6
7	7	7	7	7	7	7	7
3	6	4	6	3	6	4	6
7	7	7	7	7	7	7	7
5	6	5	6	5	6	5	6
7	7	7	7	7	7	7	7

▲図 4.1　Adam7 の各パスが担当するピクセル（8×8 ブロック内）



▲図 4.2　パスを重ねるごとに送信済みピクセルが増えていく様子

パス 1 だけで全体の 64 分の 1 のデータが届き、画像の概形がわかります。パスを重ねるごとに届いたピクセルの数が倍になっていき、パス 7 で全ピクセルがそろいます。

便利な機能ですが、ピクセルの並びを分断するためフィルターの予測が効きにくく

なり、ファイルサイズは通常少し大きくなります。高速回線が普及した現在では、使われる場面は限られています。

4.4 フィルタリング

フィルタリングは、Deflate 圧縮の前に画像データの偏りを作り出す前処理です。画像の各行について、行の中の各バイトを周囲のバイトから予測し、予測値との差分に置き換えます。

4.4.1 参照する近傍

フィルターで予測に使うのは、対象のバイトから見て左（a）、上（b）、左上（c）の三つのバイトです。画像の左端や上端では、存在しない近傍は 0 として扱います。上の行がどのフィルターで変換されていても、a、b、c には変換前の元の値を使います。

▼リスト 4.1　フィルタリングで参照する近傍

```
c   b
a   x
```

フィルターはピクセルではなくバイト単位で予測します。たとえば、トゥルーカラー画像では 1 ピクセルが 3 バイトからなるため、3 バイト前のピクセルと同じ色成分のバイトを参照します。

4.4.2 5 種類のフィルター

PNG では、近傍から予測値を求める方法に応じて、5 種類のフィルターが定義されています。

▼表 4.1　PNG のフィルター種別

番号	種別	予測値
0	None	予測しない（元の値をそのまま記録する）
1	Sub	左のバイト a
2	Up	上のバイト b
3	Average	a と b の平均（小数点以下は切り捨て）
4	Paeth	a、b、c のうち、$a+b-c$ にもっとも近いもの

None は何もしません。インデックスカラーのように、値の大小に意味がないデー

第 5 章

アニメーション PNG

PNG には APNG と呼ばれる、アニメーションのための拡張があります。2025 年の第 3 版で標準機能になりました。

本章では、APNG が生まれた経緯、APNG を構成する三つのチャンクとフレームの合成規則、そして GIF や WebP のアニメーションとの違いを解説します。

5.1 アニメーション PNG とは

GIF の代替として生まれた PNG に、長らくなかった機能がアニメーションです。

PNG は当初からアニメーションを扱わず、その役割は MNG（Multiple-image Network Graphics）という別のフォーマットが担いました。しかし MNG は仕様が大きく複雑で、デコーダーも大きくなりすぎたため、ブラウザーへの普及には至りませんでした。

2004 年、Mozilla の Stuart Parmenter 氏と Vladimir Vukićević 氏は、PNG にチャンクを三つ追加してアニメーションを実現する APNG（Animated PNG）を提案しました。しかし PNG の仕様を保守する PNG 開発グループは 2007 年 4 月の投票で、これを公式の拡張として採用しないと決めました。反対の理由には、PNG は単一画像のフォーマットだという立場のほかに、APNG が通常の PNG と同じ拡張子と MIME タイプを使うため、静止画と区別できないという指摘がありました。それでも 2008 年に Firefox 3 と Opera 9.5 が APNG のサポートを開始すると、2014 年の Safari 8、2017 年の Chrome 59 と対応が広がり、APNG は事実上の標準になりました。

そして 2025 年、PNG の第 3 版で APNG が標準機能として採用されます。

5.2 三つのチャンク

APNG は、通常の PNG チャンクに加えて、acTL、fcTL、fdAT の三つの補助チャンクで構成されます。

いずれも小文字で始まる補助チャンクなので、APNG を知らないデコーダーはこれらを無視し、IDAT の静止画だけを表示します。

2 文字目も小文字で、プライベートチャンクと同じ形式です。標準化前から広く使われていた名前を変えられなかったため、公式のチャンクながら例外的にプライベートチャンクの形式を使うことが仕様に明記されています。

5.2.1 acTL（アニメーション制御）

acTL はアニメーション全体の情報を持つチャンクです。最初の IDAT より前に、一つだけ置きます。拡張子や MIME タイプだけでは判断できないので、ファイルが APNG かどうかは、チャンクを読んで acTL の有無で見分けます。

▼表 5.1　acTL チャンクの構造

フィールド	サイズ	内容
num_frames	4 バイト	フレーム数。fcTL の数と一致させる（0 は不正）
num_plays	4 バイト	再生回数。0 なら無限に繰り返す

5.2.2 fcTL（フレーム制御）

fcTL は各フレームに一つずつ付くチャンクで、データ部は 26 バイトです。キャンバス（IHDR の幅と高さを持つ描画領域）上のフレームの位置とサイズ、表示時間、合成方法を指定します。フレームには、キャンバス全体ではなく一部の矩形領域だけを含めることもできます。前のフレームから変わった部分だけを格納すれば、ファイルが小さくなります。ただし矩形はキャンバスからはみ出せず、x_offset と width の和は IHDR の幅以下、y_offset と height の和は IHDR の高さ以下でなければなりません。後のフレームで広い領域が要るなら、IHDR の幅と高さをあらかじめ大きく取り、IDAT の静止画の余白を完全に透明な黒で埋めておきます。