PGF – Progressive Graphics File

Version 7.21.2

Christoph Stamm, University of Applied Sciences, Northwestern Switzerland

Introduction

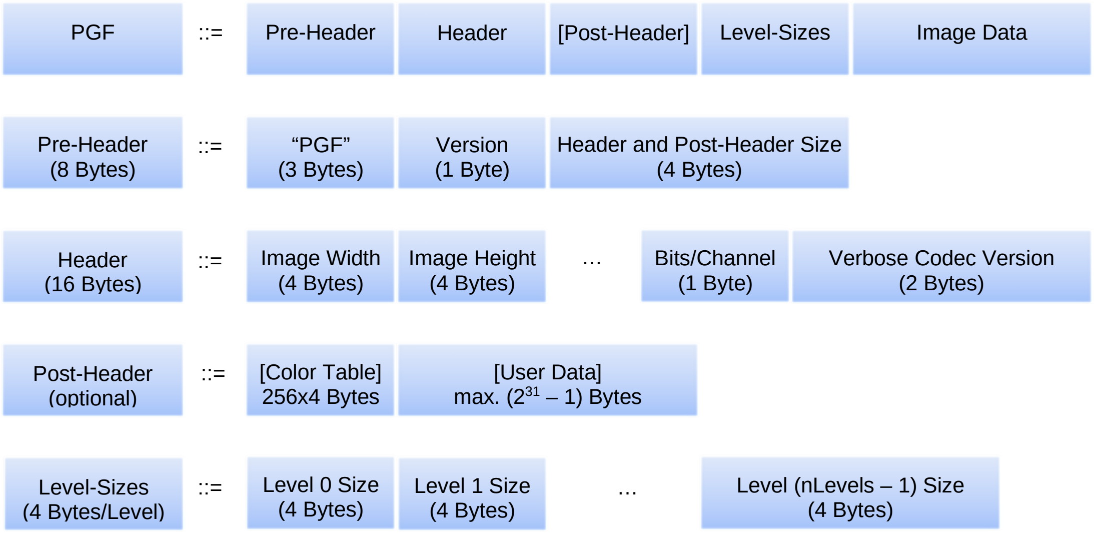
Readers of this document should be familiar with the PGF codec or should have a good understanding of image file formats, fast discrete wavelet-transform (DWT), and image compression. Introductions of PGF can be found in [Sta02, Wiki].

PGF can be compiled with one of two possible data unit sizes: 2 or 4 bytes per wavelet coefficient per channel. Codec configurations with suffix 16 use 2 bytes only per wavelet coefficient per channel. Since the wavelet transform needs 2 additional bits per coefficient the maximum number of bits per channel is either 16 – 2 = 14 or 32 – 2 = 30 bits, respectively.

In the following sections we discuss the PGF header structure of version 6 and the region of interest (ROI) coding scheme since PGF version 5.

PGF Header Structure

The following image (Fig. 1) depicts the PGF header structure in a hierarchical form:



PGF ::= Pre-Header | Header | [Post-Header] | Level-Sizes | Image Data

Pre-Header (8 Bytes) ::= "PGF" (3 Bytes) | Version (1 Byte) | Header and Post-Header Size (4 Bytes)

Header (16 Bytes) ::= Image Width (4 Bytes) | Image Height (4 Bytes) | … | Bits/Channel (1 Byte) | Verbose Codec Version (2 Bytes)

Post-Header (optional) ::= [Color Table] 256x4 Bytes | [User Data] max. (2^{31} – 1) Bytes

Level-Sizes (4 Bytes/Level) ::= Level 0 Size (4 Bytes) | Level 1 Size (4 Bytes) | … | Level (nLevels – 1) Size (4 Bytes)

Figure 1: PGF header structure

Before we start to introduce the different header fields it is worth to notice that PGF files are stored in little-endian byte order. On big-endian platforms the codec must be compiled with the __BIG_ENDIAN__ compiler directive in order to write a sound PGF image and to read it without errors.

Pre-Header

Each PGF image starts with 0x50 0x47 0x46, the ASCII code of "PGF". These magic numbers are followed by a single byte containing versioning information and a four byte unsigned integer containing the size of both the PGF header and an optional post-header.

Header

The regular header contains the full image width (UINT32) and height (UINT32), the number of levels of the DWT or resolution pyramid (UINT8), the level of compression (UINT8), the number of bits per pixel (UINT8), the number of color channels (UINT8), a constant image format mode according to Adobe's Photoshop image formats (UINT8), the used bits per channel in 16 and 32 bit per channel modes (UINT8), and a more verbose codec version (PGFVersionNumber) including major (7), year since 2000 and week in year. While most of these data fields are widely used and well known, we concentrate our further discussion on the number of pyramid levels and the level of compression (quality).

Post-Header (Color Table and Metadata)

A PGF image might also contain a so-called post-header. This (optional) post-header consists of two (optional) structures. The first structure is a mandatory color table for 8 byte indexed color images. PGF only supports 8 byte indexed color images. Hence, the color table length is 256. Each table entry is a RGB quad (blue, green, red, alpha or not used), so the total size of the color table is 1024 bytes. The second structure, called user data, is completely under control of the user or the software using the codec. You can use it for annotations, metadata e.g. EXIF information, or what else. In case you are a developer interesting in the PGF codec, you can store a total number of (2^{31} – 1) bytes of user data in a PGF image file.

In case of storing metadata we propose the following structure:

```
struct MetadataHeader {
    GUID metadataType;      // {19e4a5aa-5662-4fc5-a0c0-1758028e1057}
    UINT32 headerSize;      // size of Metadata Header in bytes
    UINT32 blockCount;      // number of used top level metadata blocks
    struct {
        GUID format;        // metadata block format
        UINT32 blockPos;    // metadata block position (relative to UserDataPos)
    } blocks[10];           // maximum 9 top level metadata blocks,
                            // blocks[blockCount].blockPos is valid and contains
};                          // the relative position of the Level-Sizes
```

This structure begins with a mandatory globally unique identifier (GUID) representing this metadata structure and allows the usage of up to 9 different top level metadata blocks. Each top level block again is identified by a GUID. The structure of a metadata block depends on the used metadata format. The metadata header size is the total size of the struct MetadataHeader: 16 + 4 + 4 + 10·(16 + 4) = 224 bytes.

The following GUIDs[1] for well-known metadata block formats should be used:

- JPEG App0: {79007028-268D-45d6-A3C2-354E6A504BC9}
- JPEG App1: {8FD3DFC3-F951-492B-817F-69C2E6D9A5B0}
- JPEG App13: {326556A2-F502-4354-9CC0-8E3F48EAF6B5}
- Exif: {1C3C4F9D-B84A-467D-9493-36CFBD59EA57}
- Gps: {7134AB8A-9351-44AD-AF62-448DB6B502EC}
- XMP: {BB5ACC38-F216-4CEC-A6C5-5F6E739763A9}
- Unknown: {A45E592F-9078-4A7C-ADB5-4EDC4FD61B1F}

Level-Sizes

The last block in Figure 1 consists of a directory of sizes of pyramid levels (called level-sizes). It is actually not part of the mandatory PGF header structure. The PGF codec doesn't use this information in a normal read/write use case with fully available data. Therefore, you usually don't need it, either. But in case you are interested in progressive decoding (reading and visualizing) of a PGF byte stream, then you should consult this directory with provided API methods. The size of the level-sizes directory depends only on the number of levels (stored in the header). Because of the fixed size of the pre-header, the stored size of header and post-header, and the computed size of the directory, you can easily calculate the number of bytes

---

[1] A list of the format GUIDs is available at http://technet.microsoft.com/en-us/query/ee719882.

needed to decode both the whole PGF header structure and the level-sizes directory in three consecutive reading steps. Before you start reading the first image level (usually a small thumbnail image) you have to make sure enough data is available for reading this level. The number of bytes needed to decode the first image level is stored in the first field of the directory. The second field contains then the number of bytes additionally needed to decode the next level and so on.

Pyramid Levels

The hierarchical DWT approach used in PGF allows a natural decomposition of the original image in a number of similar images with reduced resolutions. The original image is at level 0 and consists of four quarters (sub-bands) at level 1: LL, LH, HL, HH (see Fig. 2). The LL-sub-band, just a smaller version of the original image, is further decomposed in four quarters at level 2, and so on. The smallest image – usually thumbnail called – is in our example the LL-sub-band at level 3. In total we have four levels and each level is just a quarter of the next larger resolution. The number of levels is stored in the PGF header. Please be aware that a large level index means a small image resolution. The level index is an optional parameter of the method read() in the API. read(0) is therefore equal with reading the whole image in its original size.

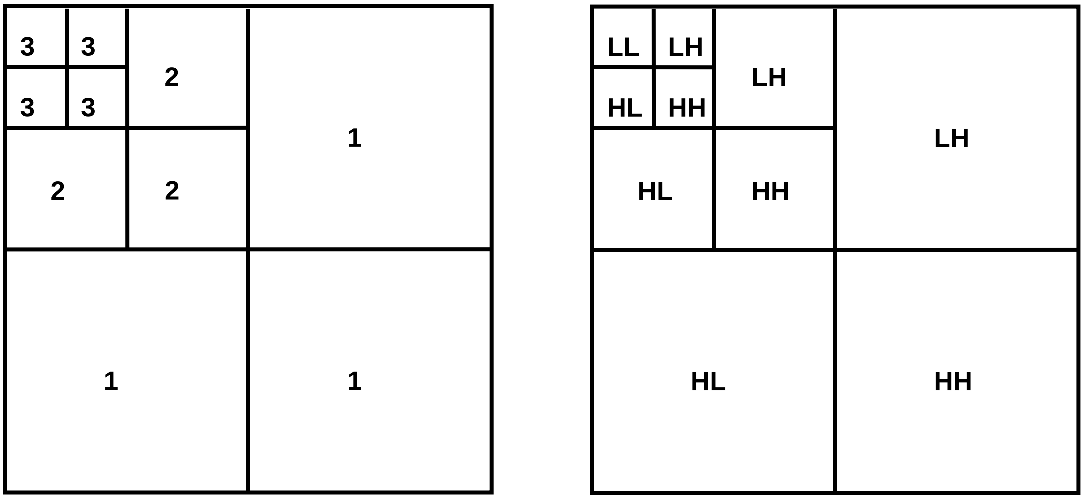


Figure 2: Resolution levels (0 - 3) and DWT pyramid (L: low pass filter, H: high pass filter)

Compression Level

In PGF terminology we call the compression level *quality*, but we use it in an inverse meaning. Usually a high quality number stands for a good or high quality, but in PGF quality 0 means lossless compression and quality 4 for example stands for lossy compression but still very good quality. Actually, 2^q is a quantization parameter for the wavelet coefficients, where q is the number stored in the quality field. It is easy to understand, that PGF encoding with a quality field of almost the number of bits per color channel results in a very poor output quality, but in a very high compression ratio.

ROI Coding Scheme

Since PGF version 5 an additional coding scheme has been introduced. The new coding scheme is only a small variation of the original one and it allows extracting of regions of interest (ROIs) without decoding the entire image. The new scheme is an addition and not a replacement because of its slightly reduced compression efficiency. The compression ratio reduction increases with increasing quality parameter; for a typical image, e.g. “hibiscus” from the Kodak test set, it is 2.3% at quality 0 (from 2.67 to 2.61), and 6.8% at quality 4 (from 13.77 to 12.83). Using the ROI coding scheme is most effective for large images where several small ROIs have to be extracted. The extraction is also progressive.

The ROI coding scheme is based on equal sized blocks, whereas the image resolution at the highest level determines the block size. Each block at level k can be independently decoded. In Figure 3 we see an example of ROI extraction. Let us assume the image size is 8000 x 8000 pixels and the number of levels is 3. So the block size is 1000 x 1000 pixels. We assume further the ROI coordinates are: top-left = (3000, 5000), bottom-right = (7000, 7000). Because the four blocks at level 3 are the smallest decoding units, the whole level 3 has to be decoded (all decoded blocks are gray shaded in Figure 3). At level 2 we only need the two lower blocks of the remaining three sub-bands LH, HL, and HH. At level 1 we already have the lower half of the LL-sub-band and from each of the three remaining sub-bands we only have to decode 6 of 16 blocks. At the end all relevant blocks have been decoded while the others have been read but not decoded.

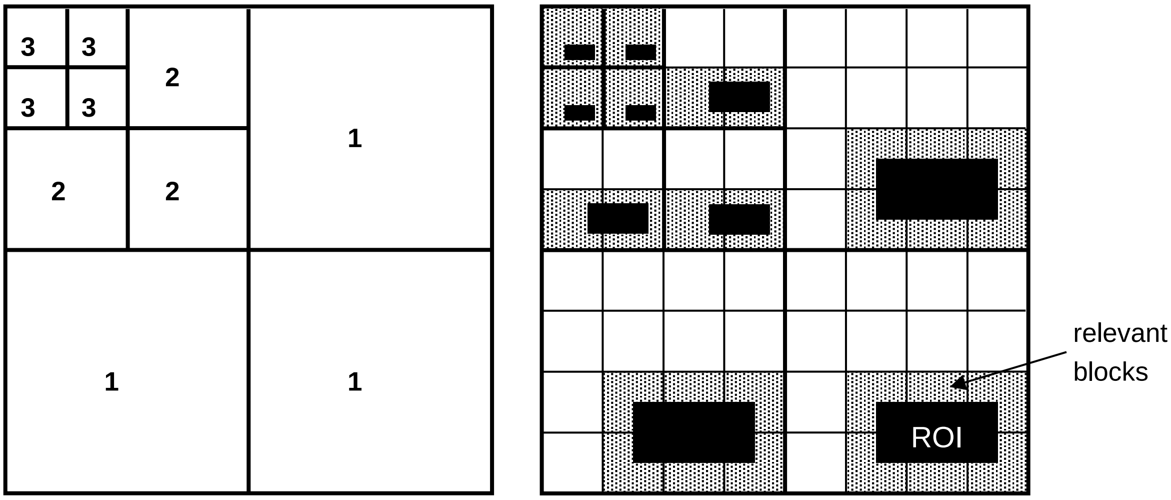


Figure 3: ROI extraction in PGF. Decoded blocks are gray shaded.

References

[Sta02] Stamm, C. PGF - A new progressive file format for lossy and lossless image compression. Journal of WSCG, Vol. 10(2), 421-428, 2002.

[Wiki] Progressive Graphics File. http://en.wikipedia.org/wiki/Progressive_Graphics_File