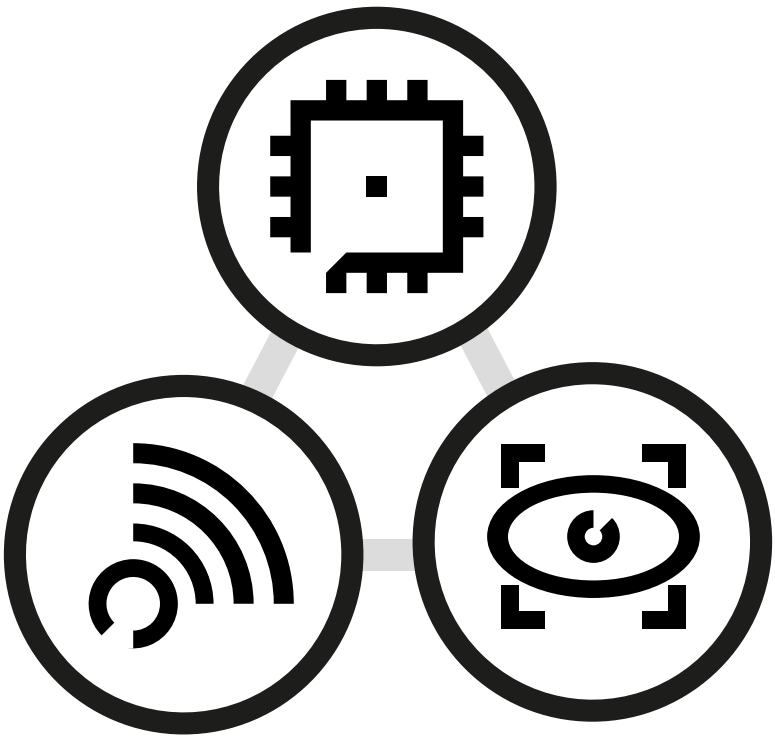
PVR File Format Specification

Revision: 1.0
11/02/2020
Public



Published: 11/02/2020-09:12

Contents

1. Introduction to PVR

An overview of the three different elements which make up PVR files

Document Overview

The purpose of this document is to act as a specification for the PVR file format (PVR specification version 3.0.0).

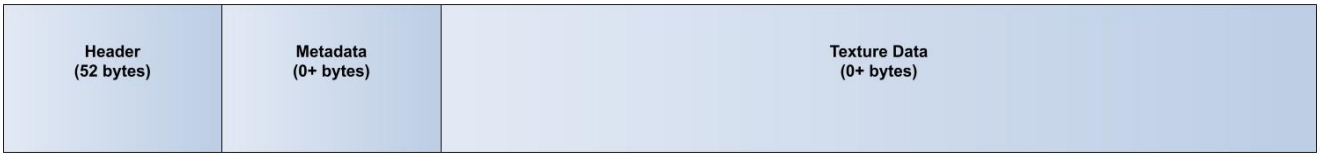
What is PVR?

PVR is a texture container format used to store textures for graphical applications.

PVR files are divided into three main three elements:

1. A header element which is 52 bytes in length
2. Zero or more metadata elements with length determined by the header
3. A texture data element with length determined by the header

The broad layout of a PVR file is shown below.



This document will go through the structure of each of these elements in detail in their own sections.

2. Header Format

A summary of the different elements which make up the header block of PVR files

The header element contains information about the data stored in a PVR file. This header information gives all of the details required to understand and process the data in the file.

This section gives a brief overview of the different sections of the header file. The block diagram below illustrates the relative sizes of each of the elements of the header.

Version (Unsigned 32-bit Integer)

Version contains the version of the PVR header format.

The exact value of *Version* will be one of the following depending upon the endianess of the file and the computer reading it:

- 0x03525650, if endianess does match.
- 0x50565203, if endianess does not match.

Flags (Unsigned 32-bit Integer)

The purpose of the *Flags* field is to allow for future proofing of the header format, giving the format the ability to specify flags that can dictate how the texture data is stored.

The currently supported flags are:

Name	Value	Description
No Flag	0	No flag has been set.
Pre-multiplied	0x02	When this flag is set, colour values within the texture have been pre-multiplied by the alpha values.

Pixel Format (Unsigned 64-bit Integer)

Pixel Format is a 64-bit unsigned integer containing the pixel format of the texture data, where the most significant 4 bytes have been set to '0' and the least significant 4 bytes will contain a 32-bit unsigned integer value identifying the pixel format.

The valid values are:

Formats	Value
PVRTC 2bpp RGB	0
PVRTC 2bpp RGBA	1

Formats	Value
PVRTC 4bpp RGB	2
PVRTC 4bpp RGBA	3
PVRTC-II 2bpp	4
PVRTC-II 4bpp	5
ETC1	6
DXT1	7
DXT2	8
DXT3	9
DXT4	10
DXT5	11
BC1	7
BC2	9
BC3	11
BC4	12
BC5	13
BC6	14
BC7	15
UYVY	16
YUY2	17
BW1bpp	18
R9G9B9E5 Shared Exponent	19
RGBG8888	20
GRGB8888	21
ETC2 RGB	22
ETC2 RGBA	23
ETC2 RGB A1	24
EAC R11	25
EAC RG11	26
ASTC_4x4	27
ASTC_5x4	28
ASTC_5x5	29
ASTC_6x5	30
ASTC_6x6	31
ASTC_8x5	32
ASTC_8x6	33
ASTC_8x8	34
ASTC_10x5	35
ASTC_10x6	36

Formats	Value
ASTC_10x8	37
ASTC_10x10	38
ASTC_12x10	39
ASTC_12x12	40
ASTC_3x3x3	41
ASTC_4x3x3	42
ASTC_4x4x3	43
ASTC_4x4x4	44
ASTC_5x4x4	45
ASTC_5x5x4	46
ASTC_5x5x5	47
ASTC_6x5x5	48
ASTC_6x6x5	49
ASTC_6x6x6	50

If the most significant 4 bytes contain a value, the full 8 bytes are used to determine the pixel format. The least significant 4 bytes contain the channel order, each byte containing a single character, or a null character if there are fewer than four channels, e.g., {'r', 'g', 'b', 'a'} or {'r', 'g', 'b', '\0'}.

The most significant 4 bytes state the bit rate for each channel in the same order, each byte containing a single 8-bit unsigned integer value, or zero if there are fewer than four channels, e.g., {8, 8, 8, 8} or {5, 6, 5, 0}.

Colour Space (Unsigned 32-bit Integer)

Colour Space is a 32-bit unsigned integer that specifies which colour space the texture data is in.

The two valid values are:

Colour Space	Value	Description
Linear RGB	0	Texture data is in the Linear RGB colour space
sRGB	1	Texture data is in the Standard RGB colour space

Channel Type (Unsigned 32-bit Integer)

Channel Type is a 32-bit unsigned integer that determines the data type of the colour channels within the texture data.

Valid values are:

Data Type	Value
Unsigned Byte Normalised	0
Signed Byte Normalised	1
Unsigned Byte	2

Data Type	Value
Signed Byte	3
Unsigned Short Normalised	4
Signed Short Normalised	5
Unsigned Short	6
Signed Short	7
Unsigned Integer Normalised	8
Signed Integer Normalised	9
Unsigned Integer	10
Signed Integer	11
Float	12

Height (Unsigned 32-bit Integer)

Height is a 32-bit unsigned integer representing the height of the texture stored in the texture data, in pixels.

Width (Unsigned 32-bit Integer)

Width is a 32-bit unsigned integer representing the width of the texture stored in the texture data, in pixels.

Depth (Unsigned 32-bit Integer)

Depth is a 32-bit unsigned integer representing the depth of the texture stored in the texture data, in pixels.

Num. Surfaces (Unsigned 32-bit Integer)

Num. Surfaces is used for texture arrays. It is a 32-bit unsigned integer representing the number of surfaces within the texture array.

Num. Faces (Unsigned 32-bit Integer)

Num. Faces is a 32-bit unsigned integer that represents the number of faces in a cube map.

MIP-Map Count (Unsigned 32-bit Integer)

MIP-Map Count is a 32-bit unsigned integer representing the number of MIP-Map levels present including the top level. A value of one, therefore, means that only the top level texture exists.

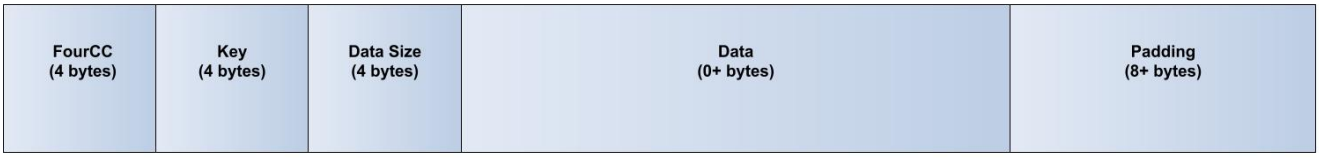
Meta Data Size (Unsigned 32-bit Integer)

Meta Data Size is a 32-bit unsigned integer representing the total size (in bytes) of all the metadata following the header.

3. Metadata Format

A summary of the different elements which make up the metadata block of PVR files

Metadata allows for the creator of a PVR file to store custom information within the PVR file relating to the storage.



FourCC (Four-byte array)

FourCC is a four-byte identifier (consisting of single byte characters or integers) whose value, combined with the value of 'Key', is used to determine how 'Data' should be handled. The values {'P', 'V', 'R', 0} to {'P', 'V', 'R', 255} (and their numerical equivalents) are reserved and must not be used except as described in this specification.

The *FourCC* metadata elements are defined as shown below.

FourCC	Key	Data Size	Data Description
'P', 'V', 'R', 3	0	Variable	An array of integers describing the position and sizes of each texture within a texture atlas. Each sequence of four integers represents the information for a single texture within the atlas and appear in the order: 1. X Position 2. Y Position 3. Width 4. Height
'P', 'V', 'R', 3	1	8	Specifies that the file contains normal map information. The 8 bytes are in the form of a 32-bit float representing the scale of the normal map, followed by a four character array describing the order of the channels, for example {'x', 'y', 'z', 'h'}. Use of 'h' as the representation for a given channel denotes that the channel in question contains the original height map.
'P', 'V', 'R', 3	2	6	Specifies that the file contains a cube map and the order of the faces within that cube map. The 6 bytes represent a six character string. This string shows the order the cube map faces are stored in the texture data, for example 'XxYyZz'. Uppercase letters refer to a positive axis position while lowercase refer to a negative axis position. Not all axes must be present.

FourCC	Key	Data Size	Data Description

'P', 'V', 'R', 3	3	3	Specifies the logical orientation of the texture within the texture data. This does not affect the mapping from pixels to texture coordinates. Each byte is a Boolean value representing the orientation for a single axis in the order X, Y, Z. The values are as follows:
			• X Axis
			  • Non-zero value = X values increase to the left
			  • Zero value = X values increase to the right
			• Y Axis
			  • Non-zero value = Y values increase upwards
			  • Zero value = Y values increase downwards
			• Z Axis
			  • Non-zero value = Z values increase outwards
			  • Zero value = Z values increase inwards

'P', 'V', 'R', 3	4	12	Specifies whether the texture has a border. The 12 bytes are broken down into three unsigned 32-bit integers. The three integers represent the size of the border of the image, in pixels, on the X, Y and Z axes, respectively. These values are used to offset texture reads by the size of the border in order to obtain the actual texture data. It should be noted that only three border sizes are given, this means that the border size for X is applied to both the left and right of the image, Y to the top and bottom and Z to the front and back.
'P', 'V', 'R', 3	5	Variable	Specifies that this block contains padding data. The size of data varies in order to align the texture data with a convenient block boundary. The contents of data are left undefined. This block should be skipped during parsing.

Key (Unsigned 32-bit integer)

Key is an unsigned 32-bit integer, which, when coupled with *FourCC* determines how *Data* should be handled.

Data Size (Unsigned 32-bit integer)

Data Size is an unsigned 32-bit integer representing the size of *Data* in bytes.

Data

Data is an array of user defined information of size determined from *Data Size* of a data type and purpose determined from the value of *FourCC* and *Key*.

Padding

Padding is a block of undefined data that can be used to ensure *Data* aligns with block boundaries. This block is not always defined and depends on the value of *FourCC*

4. Texture Data

A summary of the different elements which make up the texture data block of PVR files

The remainder of the file, after the header and metadata, is texture data. The format and size of this texture data can be found in the *header*.

Uncompressed Texture Data Structure

The uncompressed texture data is laid out as follows:

```
for each MIP-Map Level in MIP-Map Count
      for each Surface in Num. Surfaces
            for each Face in Num. Faces
                  for each Slice in Depth
                        for each Row in Height
                              for each Pixel in Width
                                    Byte data[Size_Based_On_PixelFormat]
                              end
                        end
                  end
            end
      end
end
```

Compressed Texture Data Structure

All compressed data formats have a "minimum width/height" which is the lowest number of pixels that can be represented by any given region in a compressed image.

The compressed texture data is laid out as follows:

```
for each MIP-Map Level in MIP-Map Count
      for each Surface in Num. Surfaces
            for each Face in Num. Faces
                  for each Region by aligned Depth (Based_On_PixelFormat)
                        for each Region by aligned Height (Based_On_PixelFormat)
                              for each Region by aligned Width (Based_On_PixelFormat)
                                    Byte data[Size_Based_On_PixelFormat]
                              end
                        end
                  end
            end
      end
end
```

5. Contact Details

For further support, visit our forum:

http://forum.imgtec.com

Or file a ticket in our support system:

https://pvrsupport.imgtec.com

For general enquiries, please visit our website:

http://imgtec.com/corporate/contactus.asp