Khronos Data Format Specification
1.0 release, July 2015

Andrew Garrard | Spec editor
Senior Software Engineer, Samsung Electronics
There are many ways of encoding pixel data

- **8bpp monochrome**
  
  0 1 2 3 4 5 6 7

- **32bpp red, green, blue, alpha 8888**
  
  0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7

- **32bpp blue, green, red, alpha 8888**
  
  0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7

- **16bpp red, green, blue 565**
  
  0 1 2 3 4 0 1 2 3 4 5 0 1 2 3 4

- **16bpp red, green, blue, alpha 4444**
  
  0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3
Some representations are quite complex

![Color representation](3D31_200.png)

- **YUV 4:2:0**
  - UV line interleaved
  - (8bpp layout)

- **YUV 4:2:0**
  - UV pixel interleaved
  - (8bpp layout)

...and there is a multitude of compressed texture formats

Non-color values (such as vectors) are also stored in textures and similar data can be encoded in a linear buffer or many dimensions (hence “data format”, not “pixel format”)
There are many ways to describe the “format”

- Each standard that supports more than one format needs a mechanism to select
  - And each has its own

- In Khronos, we have
  - OpenGL: GL_RGBA8
  - OpenCL: CL_RGBA + CL_UNORM_INT8
  - OpenVX: VX_DF_IMAGE_RGBX (no alpha support)

- DirectX has its own
  - D3DFMT_A8B8G8R8 (note - backwards from GL!)

- FourCC is used in AVI files (and elsewhere), and has yet another way
  - BI_RGB + 32bpp

- All these format schemes are enumerative
  - Code either understands a format or it doesn’t
  - It is impossible to write generic or future-proof code for multiple formats
  - They rely on human-readable descriptions of the formats (of varying quality)
Extra information also affects the content

- Increased gamut (source treated as encoding reduced range) - may clip
- Reduced gamut (target treated as receiving reduced range) - looks dull
- Linear transfer function (gamma) - incorrect midtones
- Encoded for NTSC displays (wrong colors)
- Also whether alpha is premultiplied, whether samples are co-sited or interpolated, etc.
- All these determine how to interpret a value just as much as bit patterns
- Typically none of this data is recorded
Lost in translation

- An API may not care about some details of the format
  - Some APIs just work on numbers and leave the interpretation to the user
  - Some APIs assume a standard internal representation and don’t care about bit layouts

- APIs often start out without support for a detail, then add it retrospectively
  - This can lead to metadata being stored separately, and may be ignored during interoperation
  - Some data gets retrospective definitions when this happens (as in FourCC BT.709)

- This is particularly a problem when APIs are chained together
  - Or when an application is communicating via an uninformed intermediate layer
Not everything is part of the "format"

- The "format" defines how to interpret elements in the data
  - The location of the elements is not part of the format

- Image-specific characteristics are separate
  - Width, height, stride, aspect ratio are all nothing to do with the format
  - Resizing an image does not change the "format"

- To access data, you need at least:
  - A pointer/reference to the start of the data for each separately-stored plane
  - If the data is multidimensional, some size and stride information

- This information is typically already application-specific
  - It is common for hardware to use tiling rather than treating the data as linear
  - APIs may have implementation-specific requirements for alignment

- Data that has a variable size is not representable in this specification
  - UTF-8 text, compressed file formats, etc. are not suitable
  - During processing, most data is fixed-size to allow fast random access
The Khronos data format descriptor

The descriptor is an array of values containing one or more descriptor blocks...

Format descriptor
- Descriptor block (basic)
- Descriptor block (extension)
- Descriptor block (vendor extension)

Basic descriptor
- Texel block description
- First sample description
- Second sample description

...one of which offers a standard way to describe most formats

Texel block description
- Color model
- Primaries
- Xfer func
- Flags
- Dimensions

Sample description
- Channel/flags
- Bits occupied
- 4D position
- Range
Future proofing

• Applications and vendors can add private descriptor blocks
  - The size of the descriptor is stored at the start, so the descriptor can be trivially copied around without needing to understand it
  - Descriptor blocks have an identifier and a size, so unknown blocks can be skipped

• Khronos defines one “basic” descriptor block
  - This is versioned, so future variants can be distinguished
  - Most common metadata is representable
  - The size of the block varies according to the complexity of the format
    - More complex formats require more “samples”
  - Any content which does not fit in the standard block can be represented in an extension block

• The basic descriptor block allows fast, intelligent comparisons
  - Most applications don’t care if two formats are identical
  - It is more useful to tell whether the same channels are present, the byte size of the format, whether color spaces match, etc. - not check everything
Mapping to APIs

- The data format descriptors are typically tens to hundreds of bytes
  - The basic descriptor block is designed for speed of interpretation by generic code
    - The mechanism is *descriptive*, meaning that large lists of enumerations are unnecessary

- Most APIs are expected to use internal enumerated formats and have a mapping to a standard descriptor for external communication
  - A descriptor is small enough for fast processing, but still quite large to use on every entry point
  - APIs might have an array to which the user can add formats, and use indices in most entry points

- The basic descriptor is simple enough that it is easy to interpret by humans
  - Having a standard format description scheme offers the promise of making documentation simpler, more complete and less ambiguous
Summary

- The Khronos Data Format Specification offers a standard, portable and flexible way to describe bulk data, such as pixel content
- Existing means of doing this are proprietary, inextensible and incomplete
- This standard representation described in this specification simplifies software, increases future-proofing, improves interoperability and facilitates documentation

- More Information
  - [www.khronos.org/dataformat](http://www.khronos.org/dataformat)