39899 Balentine Drive, Suite 125
Newark, CA 94560

Phone: 510 651 5122
Fax: 510 651 5127
URL: www.vesa.org

VESA Multiple Projector Common Data Interchange (MPCDI) Standard

Version 2.0

19 May, 2015

Purpose

The purpose of this document is to capture the definition of a data interface for components responsible for blending of graphical content across multiple displays.

Summary

This document defines a data format that is used to describe geometric warp and/or intensity blended displays. The data defines how multiple displays, typically projectors, are combined to create a single seamless display. The file format described by the Standard defines a set of warping and blending data along with a descriptor XML file bundled as a ZIP compressed archive.

Contents

Tables

Figures

Preface

Intellectual Property

Trademarks

VESA is a registered trademark of the Video Electronics Standards Association.

All other trademarks used within this document are the property of their respective owners.

Patents

There are currently no patent IPR inclusions within this standard. VESA® takes no position concerning the evidence, validity, and scope of this IPR.

The following holders of this IPR have assured VESA that they are willing to license the IPR on RAND terms. The statement of the holder of any noted IPR is registered with VESA.

Table 1: Patents

Holder Name	Contact Information	Claims Known
None	None	None

Attention is drawn to the possibility that some of the elements of this VESA Standard might be the subject of IPR other than those identified above. VESA shall not be held responsible for identifying any or all such IPR, and has made no inquiry into the possible existence of any such IPR.

Support for this Standard

Clarifications and application notes to support this standard may be written. To obtain the latest Standard and any support documentation, contact VESA.

If you have a product that incorporates Multiple Projector Common Data Interchange (MPCDI), ask the company that manufactured your product for assistance. If you are a manufacturer, VESA can assist you with any clarification you might require. Submit all comments or reported errors to VESA, in writing, using one of the following methods:

Fax:	510 651 5127, direct this fax to VESA Technical Support
Email:	support@vesa.org
Mail:	Technical Support Video Electronics Standards Association 39899 Balentine Drive, Suite 125 Newark, CA 94560

Acknowledgements

This document would not have been possible without the efforts of VESA's Multiple Projector Auto-Calibration (MPAC) Task Group. In particular, Table 2 lists the individuals and their companies that contributed significant time and knowledge to this version of the Standard.

Table 2: Main Contributors to *MPCDI v2.0*

Name	Company	Contribution
Robert Clodfelter	Barco, Inc.	Technical Contributor
Daniel Urquhart	Christie Digital Systems Canada, Inc.	Technical Contributor
David Swart	Christie Digital Systems Canada, Inc.	Technical Contributor
Christopher Jaynes	Mersive Technologies, Inc.	Technical Contributor
Steven Webb	Mersive Technologies, Inc.	Technical Contributor
Daniel Baker	WDI Research & Development	Organizational Contributor
Bei Yang	WDI Research & Development	Task Group Chair

Revision History

Table 3: Revision History

Date	Version	Description
June 17, 2013	1.0	Initial release of the Standard.
November 11, 2013	1.0a	Changed Section 3.5.2 to match Profile and Level definitions.
May 19, 2015	2.0	• Added color definitions. • Added color XML specifications. • Changed extension example to an LUT for distortion to correct for chromatic aberration. • Modified example XML type for *interpolationHint*. • Updated figure and table labels. • Reformatted document to VESA standard template (formatting, light grammar edits, corrected typos).

1 Introduction

Currently, multiple projector systems require the integration of many different components, or are sold as a complete system without extensible components. In the case of custom multi-projector displays, this can require the careful integration of projectors, image generators, warping boxes, media servers, splitters, and/or distribution amplifiers. Having all of these software and hardware components work together is a large software and hardware integration problem for anyone who is developing a new system or upgrading an existing system.

If hardware warping or blending boxes are used, the warp and blend itself must either be generated using a manual graphical user interface or provided by another piece of software. Since many warping boxes have different requirements and interfaces, it creates an environment where software integrating pieces together must support many different requirements and data formats. Even in the case where displays are provided as a fully integrated system, there are advantages to having system components produce or rely on a standard definition of the data that drives them.

Of particular interest for this Standard are multi-projector alignment systems that generate the data needed to combine individual display components into a single, seamless image. With a common standard, these systems can produce data that can then be consumed by a variety of devices, other programs, and displays without necessitating individual integration efforts. New hardware and software components will also have a lower barrier to entry, an effect that can stimulate progress and innovation. Rather than having to worry about integrating with multiple systems, a new piece of hardware and software only needs to integrate with the Standard.

Furthermore, having a common standard improves maintainability of existing systems. In any market, there are constantly changing technologies and new capabilities. Upgrading or replacing pieces currently will usually mean new custom software. With a common standard, pieces can be replaced interchangeably. Upgrading or replacing system components with different component becomes a much less costly task.

1.1 Standard Interchange

The intent of the Standard is to allow for the free interchange of data structures which define at a (sub)pixel level a video definition for color (both intensity and color elemental contribution) that have been altered to allow for conformance of multiple images to be displayed as a continuous single image. Included in this definition is geometric data, intensity, color, and topographic information about each of the displays. The purpose of this definition is to simplify the transition between technologies during the life cycle of a display system.

This information may be implemented in various topography definitions which are comprised of a variety of hardware or software applications. Figure 1-1 reflects the potential variations in topography.

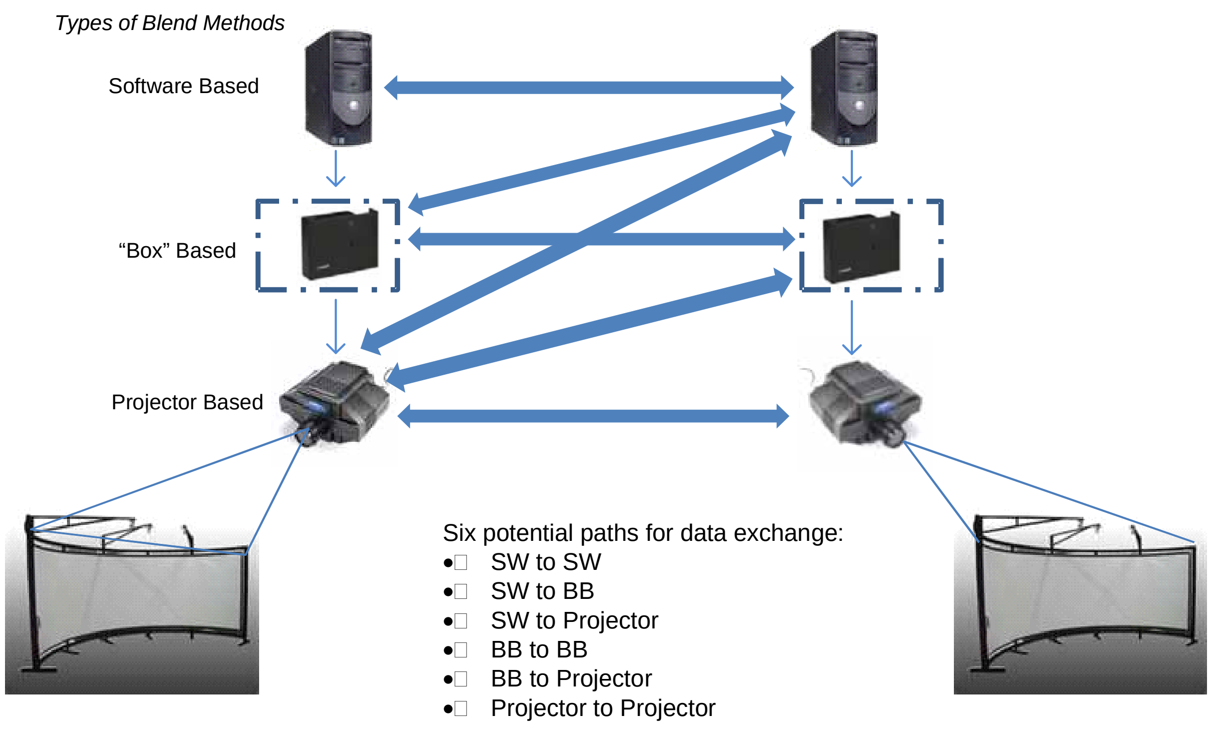

Figure 1-1: Blending Data Interchange Paths

These interfaces and the data which is to be contained within the specification takes into consideration the following fundamentals of the data required to support a warped and blended display system:

- Number of projection devices
- Resolution of projection devices/projection system
- Blend region size definition
- Area/FOV for a projection device/projection system

Other data areas are included to support the complete integration of the multiple projection display system.

1.2 Summary

Table 1-1 summarizes the MPCDI Standard.

Table 1-1: MPCDI Summary

Feature	Description
Blend Standard Descriptors	Format representing pixel blending in display space references.
Container Integration	ZIP format.
Integration Level	Standard XML interchange format.
Standardized Geometry References	2D/3D definitions in standardized format with normalized frustum definitions.
Color Data	Single component scalars or lookup tables for matching color between projectors, spatial variance within a projector, and blending upstream of the projectors.
Unique Element Management	User-defined data qualified to a specific user element.

1.3 Acronyms, Abbreviations, and Initializations

Table 1-2: Acronyms, Abbreviations, and Initializations

Acronym/Abbreviation	Stands for
BB	Blending Box
CPU	Central Processing Unit
HDR	High Dynamic Range
IEEE	Institute of Electrical and Electronics Engineers
IG	Image Generator
IPR	Intellectual Property Rights
LUT	Lookup table
MPAC	Multiple Projector Auto-Calibration (VESA) – A task group
MPCDI	Multiple Projector Common Display Interchange (VESA Standard)
PFM	Point Float Map
PNG	Portable Network Graphics
SW	Software
VESA	Video Electronics Standards Association
XML	Extended Markup Language

1.4 Glossary

Table 1-3: Glossary of Terms

Term	Definition
Alpha	Gamma-corrected intensity mapping color.
Beta	Gamma-corrected black level offset color.
Blending Box	Hardware component that provides a smoothing of multiple images into the appearance of single continuous image.
Frame Buffer (Buffer)	Entire area of a logical image. Typically the area of a bitmap.
Gamma	Non-linear operation used to encode/decode luminance in video or still image systems.
Image Generator	Image production component of a multi-display system that generates the image for a single display unit. Examples include the graphics system on a personal computer, video streaming hardware, and a single graphics pipeline on a multi-head graphics card.
Projector	Device that projects light onto a surface to show images.
Region	Area of a display (within the frame of reference of a frame buffer) covered or generated by an independent image-rendering element.
Rendering Component	Hardware or software component in a rendering system.
Rendering System	Any system that generates, warps, and/or blends images such that a final image is visible to a viewer on some screen.
Software Based	Blending methodology that relies on algorithms defined in an application that is implemented in tandem with a rendering component.
Warping	Act of taking pixels in a particular spatial location and moving them to another spatial location.
Zip File	Compressed file that is packaged using algorithms that minimize the storage size of a dataset.

1.5 Reference Documents

Table 1-4: Reference Document

Document	Version/ Revision	Referenced As	Publication Date
W3C Portable Network Graphics (PNG) Specification	2nd Edition		November 2003
VESA Intellectual Property Rights (IPR) Policy – see www.vesa.org/wp-content/uploads/2014/04/VP-200C.pdf	200C		April 14, 2014
W3C Extensible Markup Language (XML)	1.1 (2nd Edition)		September 2006
754-2008 – IEEE Standard for Floating-Point Arithmetic	2008		August 2008
About Netpbm (pfm) – see netpbm.sourceforge.net/doc/pfm.html			April 2012
PKWARE .ZIP File Format Specification	6.3.3		September 2012

2 Rendering Component Definitions

2.1 Overview

To be useful, this Standard must be capable of defining/describing the relationships between a given video generator and the full image to be displayed. Correspondingly, with the advent of high channel count projection systems and video display walls, configurations consisting of a variety of rendering components and projection methods must be supported.

A potential configuration is one in which three image generators (IGs) are creating output that will drive a multi-input projector. (See Figure 2-1.) For the geometry and intensity data for each blending box/image rendering pair to be correctly defined, the absolute position of each image renderer's un-warped image into the *frame buffer* (i.e., the X × Y space in Figure 2-1) must be specified. Figure 2-1 (left) describes the concept in pictorial form.

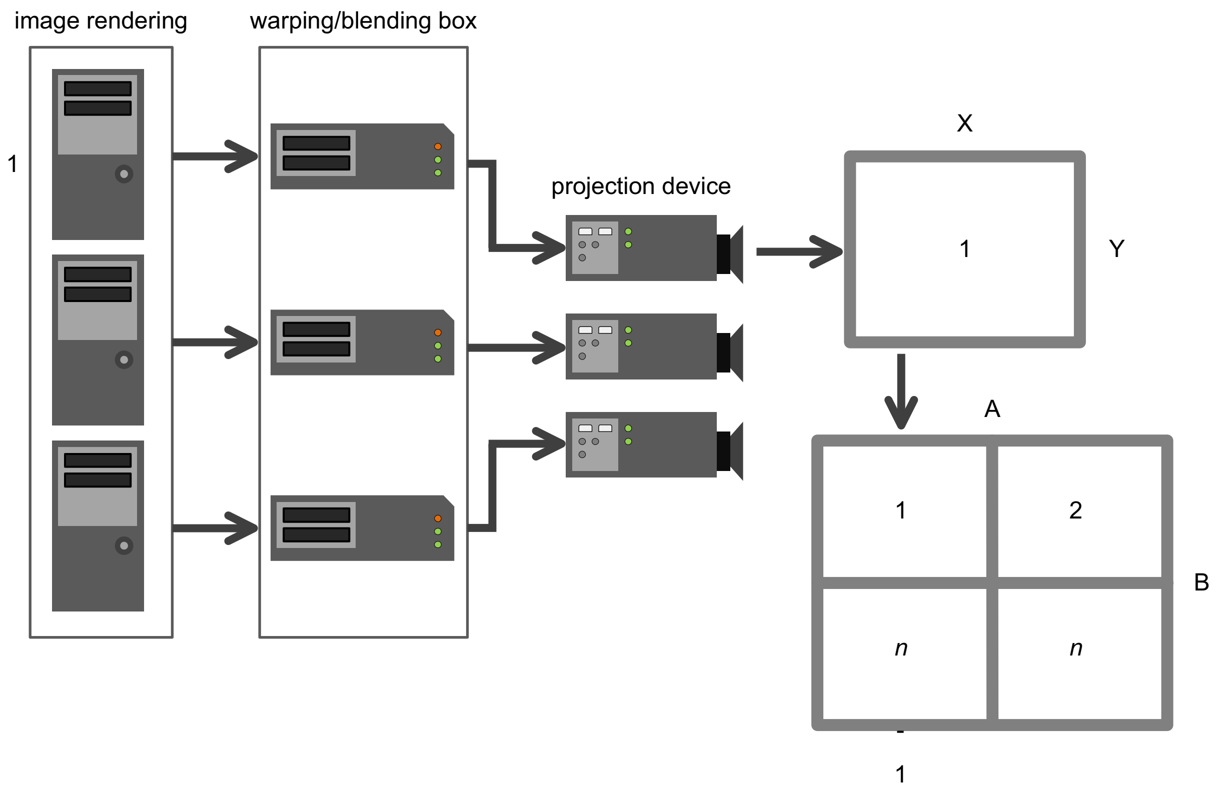

Figure 2-1: Elemental Relationships

A single projection device has a display specification (X × Y) as shown in Figure 2-1. This projection definition or frame buffer may be filled by any number (*n*) of Image Rendering paths, and typically these paths have some physical relationship to each other in the total display geometry. The individual rendering areas or *regions* are defined in terms of both their own projection area (A × B) as well as their relation to the larger display space (not shown).

Under this structure, an Image Rendering/Blending Box pair specifies a *region*, and a region definition along with a Projection Device forms a *rendering system*.

In this example, the blending definition as well as the warping modulation of the pixels projected by the regions must be capable of being defined within the overall rendering system display space. This is due to the fact that, between different regions, there should be a sharp pixel edge (i.e., no blending) as opposed to a case where blending may need to be applied when using two separate projection devices.

A different case that may be supported is one in which a single image renderer produces a 2D desktop that is scanned out to three different frame buffers (one per projector) which must be aligned. In this case, a single multi-head graphics card may act as three image generators, each producing a different frame buffer. The region data must be defined with respect to each of these frame buffers, so that output images can refer to pixels in the large region which may be outside of the frame buffer. In the case of the 2D example, a 2D offset and width/height of the region with respect to each frame buffer is sufficient.

The container definitions for warping and blending accommodate this by defining both frame buffer reference space as well as region reference space and the relationship between them for the system.

2.2 Viewing and Projection Transforms

In the case where the display system is used to render 3D scene data that is correctly warped for a particular eye point, the Standard defines how each display's viewing frustum maps to a world coordinate system. This viewing definition shall represent a right-handed coordinate system. The viewing matrix shall be used to define the per-buffer region mapping into the full display in the 3D case. The viewing matrix is defined based on yaw, pitch, and roll. When performing rotations, it is important to specify the order of rotations. Yaw shall always be done first, followed by pitch, followed by roll.

The depiction of this definition in a correlating axis configuration is in Figure 2-2.

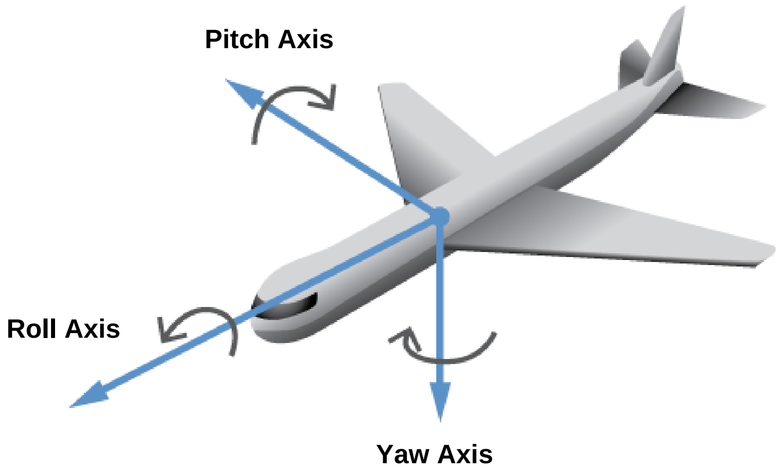

Figure 2-2: Representation of Rotations within the Viewing Matrix

2.2.1 Frustum Definition

The definition of the geometry reference for a given rendering element shall be provided to define the frustum information in a coordinate system free method. These definitions shall be provided for each viewing configuration utilizing double precision numbers for maximum resolution employed.

A Frustum is a view pyramid used in an image generator to describe a perspective view in a virtual scene. For the purposes of common geometric definition, the Frustum is made up of the seven variables listed in Table 2-1.

Table 2-1: Frustum View Matrix Definitions

Variable	XML	Definition	Unit
Yaw	*yaw*	Viewer rotates his head to the right.	Degrees
Pitch	*pitch*	Viewer rotates his head upward.	Degrees
Roll	*roll*	Viewer rotates his head clockwise.	Degrees
Right Angle	*rightAngle*	Field of View to the Right. Positive angles mean see more to the right.	Degrees
Left Angle	*leftAngle*	Field of View to the Left. Negative angles mean see more to the left.	Degrees
Up Angle	*upAngle*	Field of View up. Positive angles mean see more upward.	Degrees
Down Angle	*downAngle*	Field of View down. Negative angles mean see more downward.	Degrees

Note: *Left Angle must be less than Right Angle. Down Angle must be less than Up Angle.*

To form a matrix *RotationMatrix* to describe the frustum:

RotationMatrix(From Viewer to World) = Roll(angle) × Pitch(angle) × Yaw(angle)

where:

- Roll(angle) is a function that creates a matrix that performs roll rotation
- Pitch(angle) is a function that creates a matrix that performs pitch rotation
- Yaw(angle) is a function that creates a matrix that performs yaw rotation

The inverse of the *RotationMatrix* is:

RotationMatrix(From World to Viewer) = Yaw(-angle) × Pitch(-angle) × Roll(-angle)

To implement this rotation in standard method, the coding example on the next page is provided as an aid to users of this Standard. This algorithm is for use in applications that use Open GL.

The variables near and far relate to the near and far clipping planes set by the programmer. Here, the Z-axis is straight ahead. The Y-axis is downward. The X-axis is to the right. Most users will need to check the angles of rotation, the ordering of rotations, and whether they need the matrix below to inverse.

Note: *The following code is provided for example purposes only and is* ***not*** *guaranteed to work.*

```
double DegreesToRad = 3.14159/180.0;
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(near*tan(DegreesToRad*Frustum.LeftAngle),
          near*tan(DegreesToRad*Frustum.RightAngle),
          near*tan(DegreesToRad*Frustum.BottomAngle),
          near*tan(DegreesToRad*Frustum.TopAngle),
          near,far);
glRotated(Frustum.Yaw, 0.0, 1.0, 0.0);
glRotated(Frustum.Pitch, 1.0, 0.0, 0.0);
glRotated(Frustum.Roll, 0.0, 0.0, 1.0);
```

2.2.2 Coordinate Frame

Additional information is needed in addition to frustum definitions described in Section 2.2.1 for shader lamp implementations. Because shader lamps require a full 3D geometry scan for every projector, the full 3D position and rotation of every projector must also be provided. Thus, a coordinate frame along with a position must be provided for each shader lamp projector. This can be expressed as three vectors and a position. (Three vectors are necessary to resolve any left-handed and/or right-handed ambiguity.)

Positions can simply be expressed as a 3D point. Each of the 3D vectors relates to the Section 2.2 definitions provided for yaw, pitch, and roll. They will be expressed as orthonormal unit vectors for each of the yaw, pitch, and roll directions. (See Figure 2-2 for a graphical representation of these unit vectors.)

2.3 Gamma

Gamma correction or simply gamma is the non-linear operation used to encode/decode luminance in video or still image systems. This is used in projection blending when decoding blending values for a particular display device. Maps that are used to attenuate a display's output value can be stored in linear values or have a gamma function already applied to it. The gamma function is described as follows:

$$P_g = P_l^{\gamma}$$

where:

- P_l is the linear value
- P_g is the gamma-mapped value with γ gamma

2.4 Alpha and Beta Mapping

Intensity blending and black level adjustments in projector overlap regions are expressed in alpha and beta maps respectively. These are expressed with the following equation.

$$P_f = (P_i \times \alpha \times (1 - \beta)) + \beta$$

where:

- P_i is the initial color
- P_f is final pixel color
- α is the gamma-corrected intensity mapping color
- β is the gamma-corrected black level offset color

Note: *All variables listed above are within the range of 0.0 to 1.0.*

2.5 Color Correction

Color correction is used in multi-projector systems to achieve the following four objectives:

1 Color-match between projectors.

2 Color-match allowing for spatial variances within a single projector.

  a May also be used to compensate for a non-uniform projection surface.

3 Enabling blending upstream of the projector by characterizing the projector's response function.

4 Color lookup tables (LUTs) can also be used to achieve artistic color grading within the system.

2.5.1 Color Processing

To process an image signal to correct for color, the following stages shall be used. Note that artistic effects as described in (4) in Section 2.5, Stages 1 and 2 may be modified. Stage 3 is reserved for accurate models of the display device, used when processing data upstream of the display.

Table 2-2: Color Processing Stages

Stage		Description
1	Decode Convert to linear space	Defines how the input signal should be mapped to a linear RGB color space suitable for further processing.
2	Correct Color Apply color correction	Color matching is best done in linear space.
3	Encode Convert to projector input space	When processing video up-stream of the display device, the linear data must be converted back to a gamma or other color space. When processing video in a display device or working with a display device with a linear response, this step is omitted.

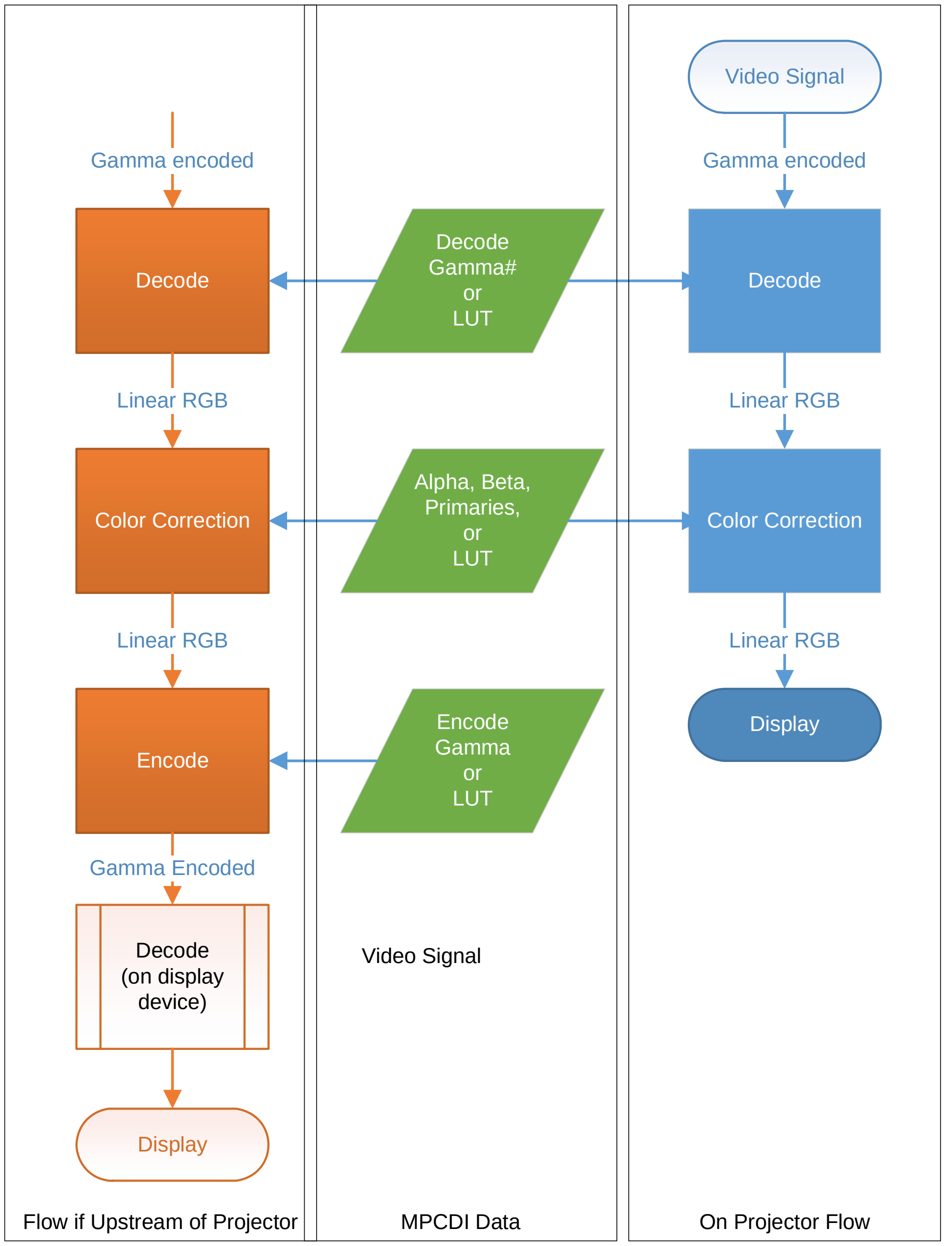

Figure 2-3: Color Processing Stages

Figure 2-3 illustrates the proposed image pipeline. **Green** parallelograms represent the three groups of captured data. **Blue** blocks indicate how the image is processed when processing is handled on a projector or for a device with a linear color response. Orange blocks indicate how the image is processed upstream of a non-linear display device.

Implementations may optimize processing by combining or re-ordering stages so long as the overall transform is mathematically equivalent.

2.5.1.1 Stage 1 – Decode

Decode defines how the input signal should be mapped to a linear RGB color space suitable for further processing.

Although this data will often be the same, it may be defined independently, per region (display). This will typically be a number. (See Section 2.5.1.2.4.) Depending on the supported color profile, an LUT may also be allowed.

2.5.1.2 Stage 2 – Correct Color

This step defines how to correct or transform the color of the image to match across the entire system, or achieve other desired effects. A 3D LUT provides almost complete freedom for global color correction or adjustment while Alpha and Beta maps allow spatial control of the white and black points. For many devices, a 3D LUT is not feasible; therefore, an alternative 1D LUT + 3x3 matrix is allowed, based on profile levels. This Standard does not describe how to generate appropriate transformations for specific applications. During this step, all processing is nominally in a linear RGB color space with a range of [0.0 to 1.0]. Figure 2-4 shows the breakout of the individual color correction steps.

For all color LUTs, an optional interpolation hint can be used and is interpreted in exactly the same way as geometry data as described in, with the exception that *keystone* may not be used as this is meaningless for color.

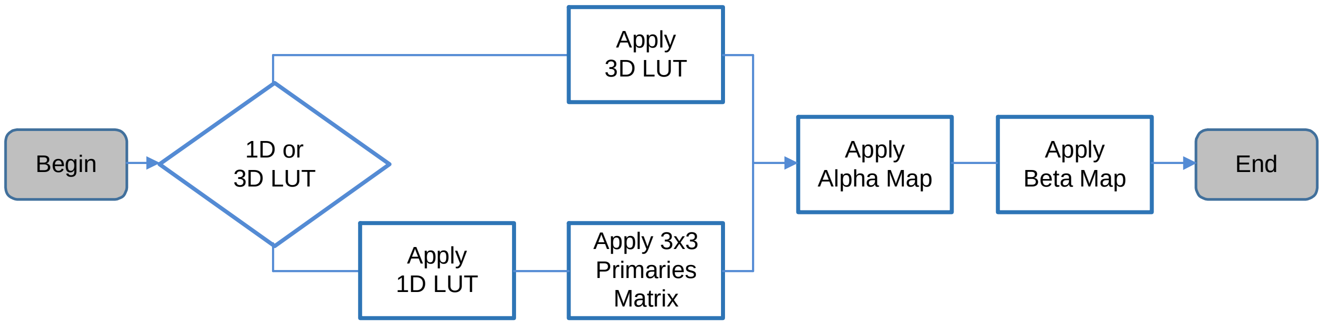

Figure 2-4: Correct Color Detail

2.5.1.2.1 RGB 1D LUT

RGB 1D LUT allows the gamma response and white-point to be adjusted globally.

2.5.1.2.2 Correction RGB Primaries

RGB primaries allow the primary colors to be shifted globally. These may be used with or without the RGB 1D LUT, but are replaced by the RGB 3D LUT. They are stored as a 3x3 matrix. This matrix should be normalized such that outputs do not exceed 100%.

Input to this transform is nominally linear; it is applied *after* the 1D LUT.

2.5.1.2.3 Correction RGB 3D LUT

RGB 3D LUT allows full control of the color response function. The RGB 3D LUT replaces the RGB primaries and RGB 1D LUT values. If both are specified, only the RGB 3D LUT is processed, where supported.

2.5.1.2.4 Gamma Definition

When a gamma function is specified, the function is given by:

$\text{Linear} = (\text{Encoded} / \text{Scale})^{\text{gamma}}$

and inverted by:

$\text{Encoded} = \text{Scale} \times \text{Linear}^{(1 / \text{gamma})}$

where:

- Linear is the linear intensity value in the range [0.0 to 1.0]
- Scale is the maximum encoded value (e.g., 255 for a typical 8-bit signal)
- Encoded is the gamma-encoded or compressed data in the range 0 to [scale]

In the case that the encoded values have an offset, this offset be subtracted from both the encoded value and the scale (i.e., Encoded = InputValue - Offset and Scale = MaxValue - Offset).

2.5.1.3 Stage 3 – Encode

The encode data defines how the display device will respond to a signal and in particular is used to encode the linear data for the display device. This data should characterize the display device as accurately as possible; otherwise blending will not be accurate.

This step is not performed when the previous step (Stage 2 – Correct Color) is executed inside of a display device, or the display device itself is inherently linear. For consistency, this value is not optional; as a best practice, it should be set to the same gamma value at *decode* when not in use.

Although this will often be the same, it may be defined independently, per region. Depending on the color level, this may be restricted to a simple gamma number, or an LUT may be allowed.

2.5.2 Pseudo Code for Color Processing

```
//decode input R,G,B values
//Pseudocode given for simple gamma case
Def (R,G,B) decode(R, G, B, data, scale=255)
      If data is a gamma number
            Gamma = data.gamma
      R1 = (R/scale)^gamma
      G1 = (G/scale)^gamma
      B1 = (B/scale)^gamma
      Return (R1, G1, B1)
      Else data must be a 1D or 3D look up table
            Return data.lookup(R,G,B);


//decode input R,G,B values
//Pseudocode given for simple gamma case
```

```
Def (R,G,B) encode(R, G, B, data, scale=255)
      If data is a gamma number
      invGamma = 1/data.gamma
      R1 = scale * R^invGamma
      G1 = scale * G^invGamma
      B1 = scale * B^invGamma
      Return (R1, G1, B1)
      Else data must be a 1D or 3D look up table
            Return data.lookup(R,G,B);

//Apply Color Correction, given 3D LUT
Def (R,G,B) correctColor3D(R, G, B, 3D_Correction_LUT)
      Return 3D_Correction_LUT[R, G, B]

//Apply Color Correction, given 1D LUT & Primaries
Def (R,G,B) correctColor1D(R, G, B, correctionData)
      //apply 1D lookup
      1D_Correction_LUT = correctionData.1D_Correction_LUT
      (R,G,B) 1D_Correction_LUT[R, G, B]
      //apply correction to primary colors
      p = correctionData.PRIMARIES //see definition of PRIMARIES in XML
      Rout = R*p.redPrimary.r + G*p.redPrimary.g + B*p.redPrimary.b
      Gout = R*p.greenPrimary.r+ G*p.greenPrimary.g+ B*p.greenPrimary.b
      Bout = R*p.bluePrimary.r + G*p.bluePrimary.g + B*p.bluePrimary.b
      Return (Rout,Gout,Bout)

// process pixel process an entire pixel through all color correction
// the pixel is not shifted in space (warped)
// R,G,B = input pixel values
// x,y = pixel coordinates, needed for alpha/beta value lookup
// decodeData data defined in "Decode" section of this document
// encodeData data defined in "Encode" section of this document
// colorCorrectionData data defined in "Correct" section above
// alpha/beta alpha and beta maps (spatial white/black points)
```

```
Def processPixel(R,G,B, x, y, decodeData, colorCorrectionData, alpha,
beta, encodeData)
      //Decode
      (R,G,B) = decode(R, G, B, decodeData)

      //Color Correct (global)
      If(colorCorrectionData is a 3D_LUT)
            (R,G,B) = correctColor3D (R, G, B, colorCorrectionData)
      Else If(colorCorrectionData is a 1D_LUT and Primaries)
            (R,G,B) = correctColor1D (R, G, B, colorCorrectionData)

      //Color Correct (local / Blending)
      //note alpha/beta may be monochrome or RGB

      R = (alpha.at(x.y).r*R*(1-beta.at(x,y).r)) + beta.at(x,y).r
      G = (alpha.at(x.y).g*G*(1-beta.at(x,y).g)) + beta.at(x,y).g
      B = (alpha.at(x.y).b*B*(1-beta.at(x,y).b)) + beta.at(x,y).b

      If(running on projector)
      Display R, G, B at x,y
            Return

      Else //must be upstream of a projector
            (R,G,B) = encode(R, G, B, encodeData)
            Send R, G, B to display device at x,y
            Return
```

2.6 Lens Distortion Mapping

In the case of Shader Lamp implementations, precise pixel alignment is important. To achieve this, a lens distortion LUT must be provided. This explicit LUT can then be used to correct for any type of distortion (such as, radial or barrel distortion from the ideal pin-hole perspective lens). Distortion LUTs can be stored the same as 2D warps because the data is identical.

2.7 Geometry Grid Interpolation

Interpolation is often used when interpreting geometry warp data for projection since storing a warp for each pixel is often not possible and unnecessary. There are many methods of interpolation that can be used on any given dataset. Because of this, MPCDI supports the following interpolation schemes within its descriptors to describe how the geometry grid data was generated and which interpolation method should be used when applying the warp grid.

- **Linear** – Bilinear piece-wise interpolation
- **Keystone** – Specifies a projective interpolation, likely only useful for a 2x2 control point grid
- **Smooth** – Any curved interpolation scheme such as (but not limited to) a bicubic interpolation
- **Unknown** – Any other interpolation method that is not strictly supported within this Standard

3 Specifications

This section specifies the MPCDI file format.

3.1 Filename

MPCDI files shall adhere to standard naming conventions of target systems. The file extension shall be .mpcdi.

[filename].mpcdi

3.2 File Format Description

The data file format shall utilize standard Extended Markup Language (XML) structures that contain specific containers and descriptors that provide for definition to determine the elements of the blending data, as listed in Table 3-1. (See also Figure 3-1.)

Table 3-1: XML Elements of Blending Data

Element	Description
Level of Integration Definition	XML descriptor file with optional frustum definitions. This will contain a profile number, self-identifier, and table of contents. If a view frustum is required for the particular profile, the frustum will be provided in the XML as well.
Geometry and Warp Definition	Defines the geometry warp that needs to take place. Data shall be stored as 32-bit floating point numbers.
Blend Maps	One or more blend maps in the form of an alpha. An alpha shall be provided to blend the overlap regions.
Optional beta offset map(s)	May be used when a display requires a per-pixel intensity offset applied to the frame buffer.
Custom Definitions as determined	To make the format extensible, optional components are to be stored within the container. Content descriptors that describe the optional components must be defined in the XML descriptor file.

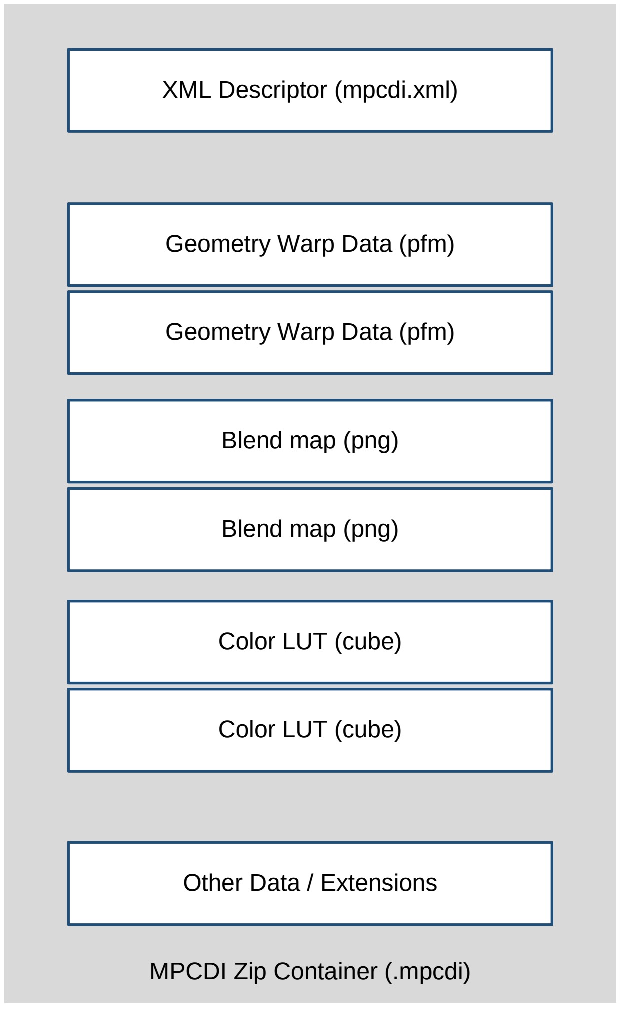

Figure 3-1: Container Format Construct

3.3 Container Format

The file format shall be a container for the associated data that will be needed for warping and blending to occur. The container itself shall be a ZIP archive. While this is a ZIP file, the file extension shall be .mpcdi. Each component within the container will be an individual file. These files are discussed in detail later in this section. All container files will be managed at the root directory level the within the container. There shall be no sub-directories within the container itself.

3.4 ZIP Format

A ZIP compression format shall be used as the container for data transport. This form of compression offers an open license and provides adequate compression at low CPU consumption. The ZIP format already has wide integration into most systems, which utilizes a form of data compression.

3.5 Profiles and Levels

Multi-display systems are used in many applications including simulation, entertainment, and corporate presentation spaces. Each of these systems has different requirements. Thus, different types and resolutions of data will need to be defined. Furthermore, different devices that implement the Standard will have different fidelity capabilities. These will be represented as profile definitions and levels. A profile defines the data's main use. The level dictates the profile's fidelity upper bound. Separate levels are used for geometry and color capabilities.

Profiles and levels are established to define the expectations of data structures for the varying integration levels that are possible within a multiple projector display system. Table 3-2 defines the profile and geometry levels. Table 3-3 defines the color levels.

Table 3-2: Profiles and Geometry Levels

Profile Name	XML	Geometry Level 1	Geometry Level 2
2D Media	2d	• One or more projectors • Warp grid not greater than 32x32	• Up to per-pixel warp resolution
3D Simulation	3d	• One or more projectors • Warp mesh no greater than every 10 pixels	• Up to per-pixel warp resolution
Advanced 3D Simulation	a3	• One or more projectors • 3D geometry of target surface • 3D mesh no greater than every 10 pixels	• Up to per-pixel 3D geometry
Shader Lamps	sl	• One or more projectors • Projector location and rotation given • 3D geometry of target surface • Up to per-pixel 3D geometry • Lens distortions not supported	• Up to per-pixel lens distortion map supported

Table 3-3: Color Levels

Color Level 1	Color Level 2	Color Level 3	Color Level 4	Color Level 5
• Single-Decode Gamma Value	• Single-Decode Gamma Value	• Decode 1D LUT	• Decode 1D LUT	• Decode 1D LUT
• Single-Channel Alpha	• Single-Channel Alpha • Single-Channel Beta	• Single-Channel Alpha • Single-Channel Beta • RGB 1D LUT • RGB Primaries	• RGB Alpha • RGB Beta • RGB 1D LUT • RGB Primaries	• RGB Alpha • RGB Beta • RGB 3D LUT
• Single-Encode Gamma Value	• Single-Encode Gamma Value	• Encode 1D LUT (inverse)	• Encode 1D LUT (inverse)	• Encode 3D LUT (inverse)

3.5.1 File Structure Representation

The file content is meant to describe the display topology for all projection elements to include the total geometry being managed as well as the “sub-regions” that require definition for blending. (See Figure 3-2.)

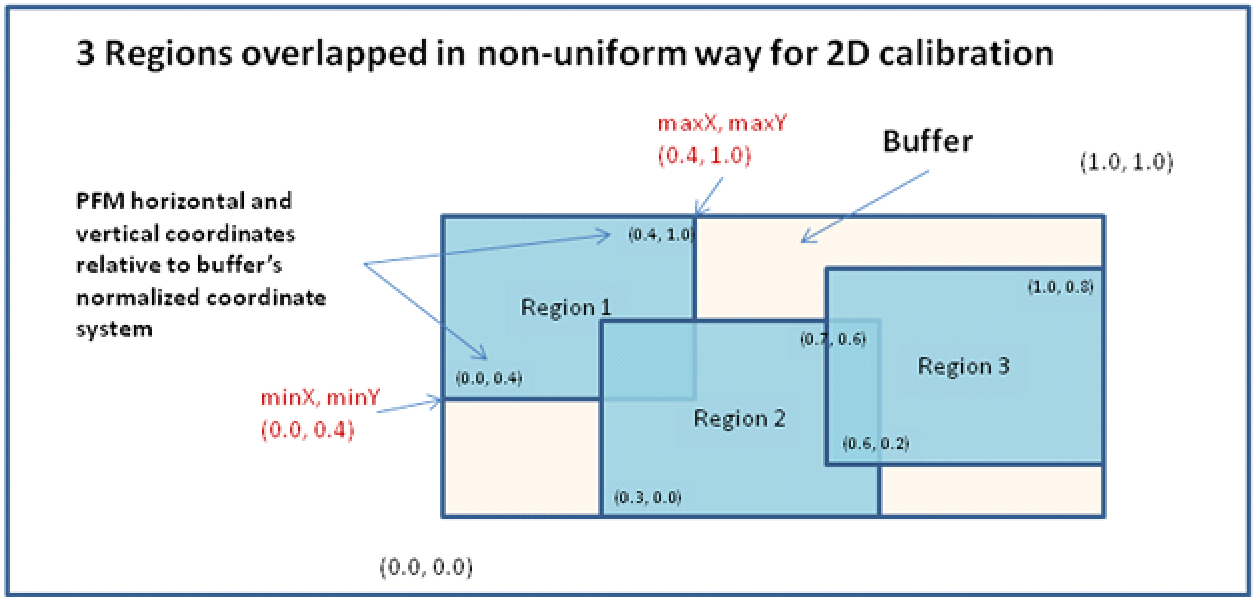

Figure 3-2: Buffer Map Diagram

The entire area of projection to be uniquely managed is contained within the overall frame buffer area definition (x, y) and the sub-areas within that frame buffer (defined as regions) represent independent projection elements within that buffer. Each of these regions contains a unique declaration of extents and resolution levels.

3.5.2 XML Descriptor

Each container shall hold one and only one XML file, named *mpcdi.xml*. The mpcdi.xml file must exist in any valid .mpcdi file. The container format shall adhere to the following structure:

- The file shall be in XML version 1.0 format with UTF-8 encoding. This means that every mpcdi.xml file shall begin with the following tag:

```
<?xml version="1.0" encoding="utf-8"?>
```

- As following with the XML conventions, tags and attribute names will be case-insensitive. However, values within attributes and tags will be treated as case-sensitive. This is particularly important because the string values that will be stored equate to filenames.
- Directly following the XML version tag shall be the main *MPCDI* tag. This tag must contain the following attributes:

```
<MPCDI profile="[valid profile name*]" geometry="[valid geometry
level*] color="[valid color level]" date="[file generation date
and time]" version="2.0">
```

Attribute	Requirements/Details
profile	Must be strings of *2d*, *3d*, *a3*, or *sl*. (See Table 3-2.)
geometry	Must be an integer from *1* to *2*. (See Table 3-2.)
color	Must be an integer from *1* to *5*. (See Table 3-3.)
date	Stored as a string in ISO 8601 format (`yyyy-MM-dd HH:mm:ss`).
version	Must be set to the string *2.0* for this particular MPCDI version.

- For an *MPCDI* tag to be complete, it must contain one display and one files section, denoted by the *display* and *files* tags listed below. After files, an *extensionSet* tag can optionally be included to provide MPCDI extensions.

Tag	Description
display	Contains information about each display. (See the next bullet for further details.)
files	Lists the files that are stored within this container format. Certain files will be required to be present depending on the profile and level. There shall be only one *files* tag per *MPCDI* tag. (See Section 3.5.)
extensionSet (Optional)	Accounts for unofficial future additions to the MPCDI Standard. (See Section 3.10 for details.)

- There is only one *display* tag, and it shall contain one or more *buffer* tag(s), and no other tags. In the case where the *profile* attribute of the *MPCDI* tag is set to *sl*, there can only be **one** *region* tag within each *buffer* tag.

- Each *buffer* tag must contain an *id* attribute and may contain optional *xResolution* and *yResolution* attributes.

Tag	Attribute	Description
buffer	*id*	Contains a string that specifies a unique identifier for this projection area. ID strings do not have to be consecutive numbers; however, they must be unique.
	xResolution (Optional)	Integer attribute that specifies (in pixels) the target display's horizontal resolution.
	yResolution (Optional)	Integer attribute that specifies (in pixels) the target display's vertical resolution.

- Each *buffer* tag must contain one or more *region* tags.

Tag	Attribute	Description
region	*id*	Contains a unique string that identifies the area of projection being described.
	x	Contains a floating point number specifying where the region starts in the X-axis. This is a number based on the total normalized coordinate in the buffer's space.
	y	Contains a floating point number specifying where the region starts in the Y-axis. This is a number based on the total normalized coordinate in the buffer's space.
	xSize	Contains a float that dictates the size of the region in normalized coordinates.
	ySize	Contains a float that dictates the size of the region in normalized coordinates.
	xResolution	Contains an integer (in pixels) that defines the total horizontal resolution of the area being described.
	yResolution	Contains an integer (in pixels) that defines the total vertical resolution of the area being described.

- *region* tag requirements
    - For cases in which the profile is set to 2D Media (*2d*) and Advanced 3D Simulation (*a3*), the *region* tag shall ***not*** contain a *frustum* or *coordinateFrame* tag.

- If the profile is 3D Simulation (*3d*) or Shader Lamps (*sl*), the region tag must contain a *frustum* tag. A *frustum* tag shall contain the seven consecutive tags, listed below, that describe the frustum. (See Section 2.2.1.)

Tag	Tag	Description
frustum	*yaw*	Shall contain one floating point number, stored in degrees. (See Section 2.2.1 for further details.)
	pitch	
	roll	
	rightAngle	
	leftAngle	
	upAngle	
	downAngle	

- If the profile is set to Shader Lamps (*sl*), the *region* tag must contain a *coordinateFrame* tag. A *coordinateFrame* tag shall contain 12 consecutive tags, listed below.

Tag	Tag	Description
coordinateFrame	*posx*	Position-related tags. (See Section 2.2.2 for details.)
	posy	
	posz	
	yawx	Yaw vector-related tags. (See Section 2.2.2 for details.)
	yawy	
	yawz	
	pitchx	Pitch vector-related tags. (See Section 2.2.2 for details.)
	pitchy	
	pitchz	
	rollx	Roll vector-related tags. (See Section 2.2.2 for details.)
	rolly	
	rollz	

- For all profiles and color levels, there shall be a *color* tag for every *region* tag. Each *color* tag shall contain a *decode* and an *encode* tag for all color levels. A *correct* tag can also be included for Color Level 1 and Color Level 2, but **must** be included for Color Level 3 through Color Level 5. For cases in which the *correct* tag is omitted, the correct *type* attribute will be assumed to be *none*.

Tag	Tag	Description
color	*decode*	Shall have a *type* attribute with *lut1d* or *gamma* options corresponding to the correct color level specification. For cases in which a gamma is used (Color Level 1 and Color Level 2), a *gamma* attribute shall be provided containing a float of the associated gamma value. If the *type* attribute is specified as *none*, the *gamma* attribute maybe omitted and the associated gamma value will be assumed to be 1.0.
	correct	Used for Color Level 3 through Color Level 5. Can optionally be provided for Color Level 1 and Color Level 2. Shall have a *type* attribute, with options as follows: • *none* for Color Level 1 and Color Level 2 • *lut1d-primary* for Color Level 3 and Color Level 4 • *lut3d* for Color Level 5 In the case of Color Level 1 and Color Level 2 in which the *type* attribute of a *correct* tag is set to *lut1d-primary*, the *correct* tag shall contain a *primaries* tag to hold the needed color transform matrix. The *primaries* tag must contain the following three tags: • *redPrimary* • *greenPrimary* • *bluePrimary* Each of the three *primary* tags shall contain the following three tags, each of which must contain a single floating point number specifying the color transform matrix: • *red* • *green* • *blue*
	encode	Shall have a *type* attribute, with options as follows: • *none* or *gamma* for Color Level 1 and Color Level 2 • *lut1d* for Color Level 3 and Color Level 4 • *lut3d* for Color Level 5 If the *type* attribute is specified as *none*, the *gamma* attribute maybe omitted and the associated gamma value will be assumed to be 1.0.

- The *files* tag shall contain a list of all files needed for warping and blending. Each *files* tag shall also contain one or more *fileset* tags and no other tags.
    - Each *fileset* tag shall contain a *geometryWarpFile* tag and an *alphaMap* tag. A *fileset* tag must also contain a *betaMap* tag if a color level higher than 2 is specified in the *MPCDI* tag. A *distortionMap* tag must be used if the profile is set to Shader Lamps (*sl*).
    - The *fileset* tag contains a region identifier that correlates to a buffer region with which the files are correlated. File identifiers are located within the *fileset* tag.
        - Within each sub-tag of the *fileset* tag, there must be a *path* tag. The *path* tag shall contain a string containing the filename of the associated files. This applies to all tags described below. Filenames must be specified in UTF-8 format.
        - A *geometryWarpFile* tag must exist within the *fileset* tag. The *geometryWarpFile* tag shall include the following tags, as needed.

Tag	Tag	Description
geometryWarpFile	*geometricUnit*	States the unit of measure in which the warp data is provided. Valid values are *mm*, *cm*, *dm*, *m*, *in*, *ft*, *yd*, and *unknown*. Must be present inside the *geometryWarpFile* tag if the profile is set to Advanced 3D Simulation (*a3*) or Shader Lamps (*sl*).
	originOf3DData	States the origin of the 3D data. Valid values are *centerOfMass*, *idealEyePoint*, *floorCenter*, and *unknown*. Must be equal to all other *originOf3DData* tags that exist under the parent file tag. If the profile is set to Shader Lamps (*sl*), this tag ***cannot*** be set to *idealEyePoint*. Must be present inside the *geometryWarpFile* tag if the profile is set to Advanced 3D Simulation (*a3*) or Shader Lamps (*sl*).
	path	Contains the path within the container of a geometry warp file.
	interpolation	States the preferred method for interpolating between control points of a warp mesh. Valid values are *linear*, *keystone*, *smooth*, and *unknown*; as described in Section 2.7.

- An *alphaMap* tag must exist within the *fileset* tag. The *alphaMap* tag must include the *componentDepth*, *bitDepth, and gammaEmbedded* tags.

  A *betaMap* tag must also exist within the *fileset* tag if a color level higher than 2 is specified in the *MPCDI* tag. The *betaMap* tag must include the *componentDepth* and *bitDepth* tags.

Tag	Tag	Description
alphaMap *betaMap*	*bitDepth*	Indicates the number of bits that comprise the component. Valid values are *8*, *16*, and *32*.
	componentDepth	Indicates the number of components within the alpha or beta map. Valid values are *1* or *3*.
	path	Contains the path within the container of the alpha or beta map.
alphaMap	*gammaEmbedded*	Stores a float inside the tag with the associated alpha map's gamma value. Alpha with no gamma can use a value of 1.0, which would be equivalent to linear. The gamma equation is described in Section 2.3.

- A *distortionMap* tag shall exist under the *fileset* tag if the profile is set to Shader Lamps (*sl*) and the geometry level is set to 2 or higher. (See Section 3.5.) Because distortion maps are always a fixed format, only a *path* tag shall exist under the *distortionMap* tag.

Tag	Tag	Description
distortionMap	*path*	Contains the path within the container of the distortion map.

- Color LUTs may be included in the *fileset* tag, depending on the color level(s) specified in Section 3.5. Each of the three possible LUT file tags of *decodeLUT, encodeLUT,* and *correctLUT* shall each have a *path* and an *interpolation* tag within it.

Tag	Tag	Description
decodeLUT *encodeLUT* *correctLUT*	*path*	Contains the path within the container of an LUT.
	interpolation	Similar to the tag of the same name within the *geometryWarpFile* tag. States the preferred method for interpolating between control points of a warp mesh. Valid values are *linear*, *smooth*, and *unknown*, as described in Section 2.7. **Note:** *This tag does* **not** *include the keystone option.*

- The optional *extensionSet* tag can include the optional *extension* tag, which can be provided for meta information and optional additional fileset(s) to an .mpcdi file. Each *extension* tag must contain the attributes listed below.

Tag	Attribute	Requirements/Details
extension (Optional)	*name*	Unique name to assign to the extension. Can be any combination of legal UTF-8 characters.
	hasExternalFiles	Boolean value specifying whether this extension contains additional filesets.

3.5.3 XML File

Note: *Variables are denoted in square brackets [] and followed by the variable's data type.*

```
<?xml version="1.0" encoding="utf-8"?>
<MPCDI profile="[valid profile name*]" geometry="[valid geometry level*]"
color="[valid color level]" date="[file generation date and time]"
version="2.0">
  <display>
    <buffer id="[unique buffer string]" xResolution ="[int]"
    yResolution="[int]">
      <region id="[unique region string]" x="[float]" y="[float]"
      xSize="[float]" ySize="[float]" xResolution ="[int]"
      yResolution="[int]">
        <frustum>
          <yaw>[float angle degrees]</yaw>
          <pitch>[float angle degrees]</pitch>
          <roll>[float angle degrees]</roll>
          <rightAngle>[float angle degrees]</rightAngle>
          <leftAngle>[float angle degrees]</leftAngle>
          <upAngle>[float angle degrees]</upAngle>
          <downAngle>[float angle degrees]</downAngle>
        </frustum>
        <coordinateFrame>
          <posx>[float]</posx>
          <posy>[float]</posy>
          <posz>[float]</posz>
          <yawx>[float]</yawx>
          <yawy>[float]</yawy>
          <yawz>[float]</yawz>
          <pitchx>[float]</pitchx>
          <pitchy>[float]</pitchy>
          <pitchz>[float]</pitchz>
          <rollx>[float]</rollx>
          <rolly>[float]</rolly>
          <rollz>[float]</rollz>
        </coordinateFrame>
        <color>
          <decode type="[lut1d | gamma]" gamma="[float]"></decode>
          <correct type="[none | lut1d-primary | lut3d]">
            <primaries>
              <redPrimary>
                <red>[float]</red>
                <green>[float]</green>
                <blue>[float]</blue>
              </redPrimary>
              <greenPrimary>
                <red>[float]</red>
                <green>[float]</green>
                <blue>[float]</blue>
              </greenPrimary>
              <bluePrimary>
                <red>[float]</red>
                <green>[float]</green>
                <blue>[float]</blue>
              </bluePrimary>
            </primaries>
          </correct>
          <encode type="none | lut1d | lut3d | gamma" gamma="[float]">
          </encode>
        </color>
      </region>
      ...
      <region ...=""></region>
    </buffer>
    ...
    <buffer ...=""></buffer>
  </display>
  <files>
    <fileset region="[region unique string]">
```

```
        <geometryWarpFile>
          <geometricUnit>[validGeometricUnit*]</geometricUnit>
          <originOf3DData>[valid3DOrigin*]</originOf3DData>
          <path>[pathToGeometryWarpFile.pfm] string</path>
          <interpolation>[validInterpolationHintValue*]</interpolation>
        </geometryWarpFile>
        <alphaMap>
          <bitDepth>[blendMapBitDepth]</bitDepth>
          <componentDepth>[blendMapComponentDepth]</componentDepth>
          <gammaEmbedded>[gammaValueEmbedded] float</gammaEmbedded>
          <path>[pathToAlphaMapFile.png] string</path>
        </alphaMap>
        <betaMap>
          <bitDepth>[blendMapBitDepth]</bitDepth>
          <componentDepth>[blendMapComponentDepth]</componentDepth>
          <path>[pathToBetaMapFile.png] string</path>
        </betaMap>
        <distortionMap>
          <path>[pathToDistortionMap.png] string</path>
        </distortionMap>
        <decodeLUT>
          <path>[pathToDecodeLUT.cube] string</path>
          <interpolation>[validInterpolationHintValue]</interpolation>
        </decodeLUT>
        <correctLUT>
          <path>[pathToCorrectLUT.cube] string</path>
          <interpolation>[validInterpolationHintValue]</interpolation>
        </correctLUT>
        <encodeLUT>
          <path>[pathTopathToEncodeLUT.cube] string</path>
          <interpolation>[validInterpolationHintValue]</interpolation>
        </encodeLUT>
      </fileset>
      <fileset ...=""></fileset>
    </files>
    <extensionSet>
      <extension Name="[name]" hasExternalFiles="[boolean]">
        ...
      </extension>
      ...
      <extension Name="[name]" hasExternalFiles="[boolean]">
        ...
      </extension>
    </extensionSet>
  </MPCDI>
```

3.6 Geometry Warp

Geometry warping data can be either 2D data or 3D data, depending on which profile the current file supports. It will have following properties:

- Geometry warping data will be a uniform grid. Many hardware warping boxes can currently only support uniform grids. This may incur larger total file sizes than a variable sampling grid; however, this problem should be minimized by file compression.
- Because warping data is a uniform grid, the data can be stored at different resolutions. Different profile levels determine the maximum allowable size for the geometry warp grid. This allows for scalability in which pixel to warping data is 1:1 or geometry data can be at a different resolution and/or aspect ratio to the actual display device.
- To avoid gross interpolation issues in the implementation of this Standard, the uniform grid shall be in a fill layout. The top left of the first pixel will be the top left of the raster image. The bottom right of the last pixel will be the bottom right of the raster image.
- Data will be stored as signed 32-bit floating numbers (IEEE 754).
- This Standard does ***not*** define little or big endianness of floating point data storage. Endianness shall be properly stored according to the PFM format.
- Geometry warp must be a minimum of a 2x2 grid. Mappings of a 1x1 grid are ***not*** allowed.

3.6.1 Portable Float Map Format

Geometry data will be stored as a three-component (RGB) PFM file. The PFM file is a basic High Dynamic Range (HDR) image format. The file format provides a simple implementation of the storage of floating point image data, in raster scan order. The first pixel is the top left pixel of the image. The last pixel in the array is the bottom right pixel of the image. Because this Standard uses a uniform grid, geometry data maps well onto this format. This is an open format described at netpbm.sourceforge.net/doc/pfm.html.

3.6.2 2D Data

For media applications and simulation applications with fixed frustums, only a 2D mapping is needed to warp a source image onto a destination surface. 2D data will have the following properties:

- 2D data will be stored as an absolute lookup that maps a source pixel to a destination pixel.
- This mapping will be stored in forward manner in which the data stored maps source pixels coming from an image generator (IG) or media server to the destination.
- These coordinates are normalized coordinates (0.0 to 1.0).
- The X and Y coordinates of the mapping will be stored in the PFM file's R and G components, respectively.
- The last component (B) of the PFM format can optionally be used for an estimated error value. Error values will be a normalized value from 0.0 to 1.0, with 0.0 denoting that there is no error and a measure of 1.0 denotes that the data is meaningless. This value is normalized on the output X and Y coordinates. A value of 0.5 indicates that the value at that location has a potential error of half the total size. If errors are not being calculated, this component should be set to Not a Number (NaN).
- This type of lookup and storage is the same as OpenGL UV mapping.
- Locations in which data is unknown should be set to NaN, but also be associated with an error of 1.0 in the last component of the data.

3.6.3 3D Data

Full 3D data is needed for applications in which the view frustum must update every frame. 3D data will have the following properties:

- 3D data will be stored as world geometry coordinate space, as a uniform grid of arbitrary size.
- The X, Y, and Z coordinates of the mapping will be stored in the PFM file's R, G, and B components, respectively.
- Because 3D data points might be unknown, these values should be stored as NaN.
- This type of lookup and storage is the same as OpenGL UV mapping (same as for 2D Data; see Section 3.6.2).
- World space geometry will be stored in a right-handed coordinate system. It is preferred that Y is up, X is to the left, and Z is into the screen and forward direction.
- 3D data can follow enumeration of real world units – *mm*, *cm*, *dm*, *m*, *in*, *ft*, *yd*, and *unknown*. This unit will be specified in the XML descriptor file. Warp generators are encouraged to express this as a real world unit rather than unknown.
- The data origin (0, 0, 0) shall be one of the following – *centerOfMass*, *idealEyePoint*, *floorCenter*, and *unknown*. Again, it is discouraged to store data with an unknown origin.

3.7 Blend Maps

A single blend map is required for a complete file. Blend maps will have the following properties:

- Blend maps will be one file – either a single Alpha/Multiply file or a combination of an Alpha/Multiply and Beta/Offset file.
- Both of these files will be stored in the Portable Network Graphics (PNG) format.
- Either file can be comprised of one or three components (monochrome or RGB, respectively).
- Either file can be 8- or 16-bits per pixel.
- Either file can be different in image size.
- Files will be arbitrary in file size.
- Data can have a pre-applied gamma/power function. This will be specified in the XML file.
- This type of lookup and storage is the same as OpenGL UV mapping (same as for 2D Data; see Section 3.6.2).
- It is recommended that blend maps be stored in a non-dithered form.
- Blend maps must be a minimum of 2x2 pixels. 1x1 mappings are not allowed.

3.7.1 Portable Network Graphics

PNG files provide an efficient way to store pixel data. The files offer an open use as-is license. Furthermore, PNG files can offer lossless compression of any enumeration of image data.

3.8 Color Lookup Tables

Color correction data is stored as a gamma number, which is stored in the XML descriptor file or as LUTs are stored as IRIDAS .cube formats. This ASCII-based format supports both 1D and 3D LUTs and is defined at doc.iridas.com/index.php?title=LUT_Formats.

To ensure a workable file-size there are some restrictions to the color LUTs:

- 1D LUTs must have an LUT_1D_SIZE between 4 and 65536, inclusive.
- Nominal input range is defined as 0.0 to 1.0 (i.e., defining DOMAIN_MAX > 1.0 is reserved for future use with HDR data).
- Software or hardware processing of the LUTs is expected to resample, as needed, to their internal processing format. When up-sampling, it is recommended, but not required, that any interpolation hint be honored.
- Individual decimal values should be specified to no more than 10 digits, thereby providing approximately 33 bits of precision.

3.9 Distortion Maps

Distortion maps are simply LUTs that map a 2D output image to another 2D output image. The maps are stored in the exact same format as 2D Geometry Warp files and should be interpreted in the same way. (See Section 3.6.2.)

3.10 Custom File Definition

To make the file format more flexible, other files can be stored within the ZIP file as long as they are named and defined within the XML file.

3.10.1 XML Specification

The following is an example of XML structure for custom definitions. It is provided ***only*** as an example. Any valid XML can be used within the *extension* tag.

```
<extension Name="[name]" hasExternalFiles="True">
  <definitions>
    <customTag1></customTag1>
    <customTag2 id="[unique identifier]"></customTag2>
  </definitions>
  <fileSet>
    <file id="[unique identifier]">
      [filename]
    </file>
    <fileseq id="[unique identifier]" numFiles="[number of files]">
      [filename1] [filename2] ...
    </fileseq>
  </fileSet>
</extension>
```

3.10.1.1 Example 3D Distortion Lookup Table XML

The following is an example of what an extension may look like. This shows how a 3D distortion lookup can be used to solve for chromatic aberration. Given an RGB value, there is an additional offset expressed as a UV coordinate encoded within a 16-bit PNG file. A distortion maximum is provided to increase precision. The maximum offset is expressed as a percentage to the remainder of the image.

Note: *This example is provided for reference purposes only and is* ***not*** *part of the official Standard.*

```
<extension Name="aberrationLookup" hasExternalFiles="True">
  <definitions>
    <distortionMax>0.2<distortionMax>
    <distortionMap id="projector1"></distortionMap>
    <distortionMap id="projector2"></distortionMap>
  </definitions>
  <fileSet>
    <fileseq id="fw0" numFiles="16">
      abb_proj_01_00.png
      abb_proj_01_01.png
```

```
          abb_proj_01_02.png
          abb_proj_01_03.png
          abb_proj_01_04.png
          abb_proj_01_05.png
          abb_proj_01_06.png
          abb_proj_01_07.png
          abb_proj_01_08.png
          abb_proj_01_09.png
          abb_proj_01_10.png
          abb_proj_01_11.png
          abb_proj_01_12.png
          abb_proj_01_13.png
          abb_proj_01_14.png
          abb_proj_01_15.png
        </fileseq>
        <fileseq id="inv0" numFiles="16">
          abb_proj_02_00.png
          abb_proj_02_01.png
          abb_proj_02_02.png
          abb_proj_02_03.png
          abb_proj_02_04.png
          abb_proj_02_05.png
          abb_proj_02_06.png
          abb_proj_02_07.png
          abb_proj_02_08.png
          abb_proj_02_09.png
          abb_proj_02_10.png
          abb_proj_02_11.png
          abb_proj_02_12.png
          abb_proj_02_13.png
          abb_proj_02_14.png
          abb_proj_02_15.png
        </fileseq>
      </fileSet>
    </extension>
```

A Main Contribution History

Table A-1: Main Contributors to *MPCDI v1.0*

Name	Company	Contribution
Robert Clodfelter	Barco, Inc.	Technical Contributor
David Swart	Christie Digital Systems Canada, Inc.	Technical Contributor
Christopher Jaynes	Mersive Technologies, Inc.	Task Group Co-vice Chair, Technical Contributor
Samson Timoner	Scalable Display Technologies, Inc.	Task Group Co-vice Chair
Daniel Baker	WDI Research & Development	Organizational Contributor
Bei Yang	WDI Research & Development	Task Group Chair

Table A-2: Main Contributors to *MPCDI v1.0a*

Name	Company	Contribution
Robert Clodfelter	Barco, Inc.	Technical Contributor
Daniel Urquhart	Christie Digital Systems Canada, Inc.	Technical Contributor
David Swart	Christie Digital Systems Canada, Inc.	Technical Contributor
Christopher Jaynes	Mersive Technologies, Inc.	Technical Contributor
Steven Webb	Mersive Technologies, Inc.	Technical Contributor
Rajeev Surati	Scalable Display Technologies, Inc.	Technical Contributor
Samson Timoner	Scalable Display Technologies, Inc.	Technical Contributor
Daniel Baker	WDI Research & Development	Organizational Contributor
Bei Yang	WDI Research & Development	Task Group Chair