Video Electronics Standards Association

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

June 17, 2013

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.

Table of Contents

Tables

Figures

Preface

Intellectual Property

While every precaution has been taken in the preparation of this standard, the Video Electronics Standards Association and its contributors assume no responsibility for errors or omissions, and make no warranties, expressed or implied, of functionality or suitability for any purpose.

Trademarks

All trademarks used within this document are the property of their respective owners. VESA is a trademark of the Video Electronics Standards Association.

Patents

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

Attention is drawn to the possibility that some of the elements of this VESA Standard may 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.*

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.

Holder Name	Contact Information	Claims Cited
None	None	None

THIS STANDARD IS BEING OFFERED WITHOUT ANY WARRANTY WHATSOEVER, AND IN PARTICULAR, ANY WARRANTY OF NON-INFRINGEMENT IS EXPRESSLY DISCLAIMED. ANY IMPLEMENTATION OF THIS STANDARD SHALL BE MADE ENTIRELY AT THE *IMPLEMENTER'S* OWN RISK, AND NEITHER VESA, NOR ANY OF ITS *MEMBERS* OR *SUBMITTERS,* SHALL HAVE ANY LIABILITY WHATSOEVER TO ANY *IMPLEMENTER* OR THIRD PARTY FOR ANY DAMAGES OF ANY NATURE WHATSOEVER DIRECTLY OR INDIRECTLY ARISING FROM THE IMPLEMENTATION OF THIS STANDARD.

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, which incorporates this standard, you should ask the company that manufactured your product for assistance. If you are a manufacturer, VESA can assist you with any clarification you may require. Submit all comments or reported errors in writing to VESA using one of the following methods.

Fax: 510 651 5127, direct this fax to Technical Support at VESA

E-mail: support@vesa.org

Mail: Technical Support
VESA
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, the following individuals and their companies contributed significant time and knowledge to this standard document.

Table 0-1: Main Contributors to Version 1

Name	Company	Contribution
Robert Clodfelter	Barco, Inc.	Technical Contributor
David Swart	Christie Digital Systems Canada, Inc.	Technical Contributor
Christopher Jaynes	Mersive Technologies	Task Group Co-Vice Chair, Technical Contributor
John Little	*Nova Technologies*	*Editor (non-mem)*
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

Revision History

June 17, 2013 Initial release of the standard

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. The potential variations in topography are reflected in Figure 1-1.

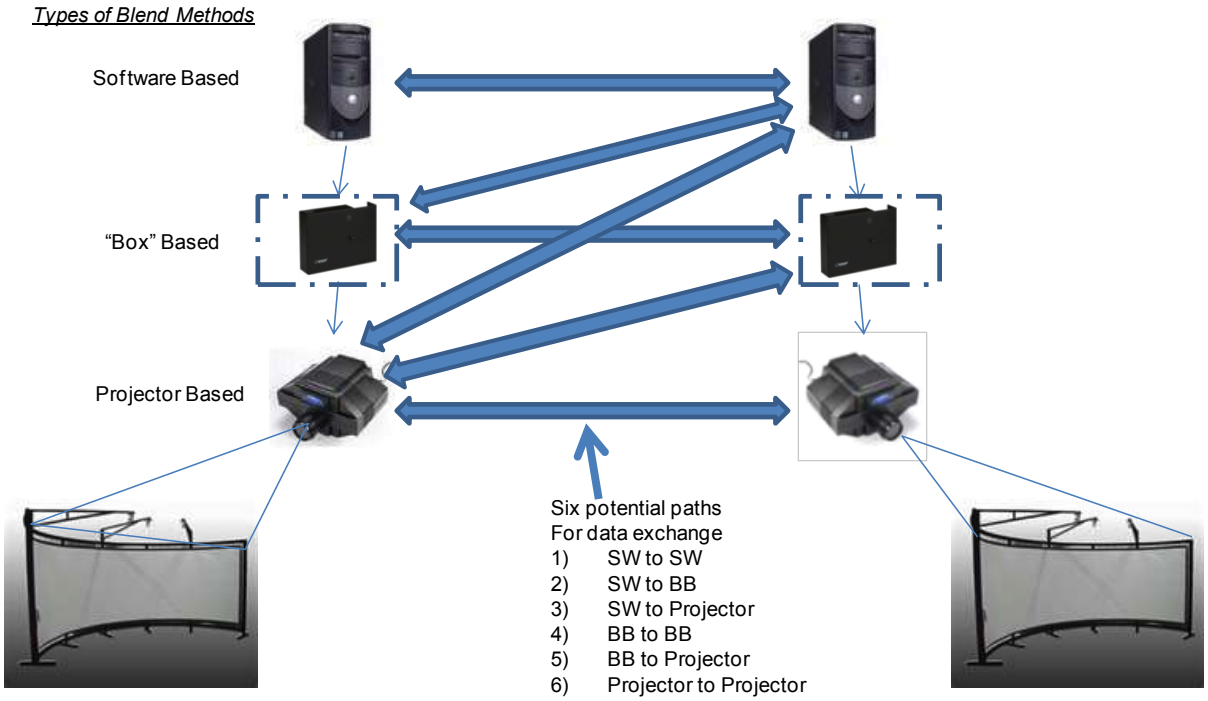

Figure 1-1: Blending Data Interchange Paths

These interfaces and the data which is to be contained within the standard 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 of MPCDI

The following table summarizes the MPCDI Standard.

Table 1-1: MPCDI Summary

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

Unique Element Management	User-defined data qualified to a specific user element

1.3 References

Table 1-2: Reference Documents

Document	Version/Revision	Date
W3C Portable Network Graphics (PNG) Specification	Second Edition	November 2003
VESA Intellectual Property Rights (IPR) policy 200 – see www.vesa.org/Policies/ipp.htm	B	February 2005
W3C Extensible Markup Language (XML)	1.1 (Second Edition)	September 2006
754-2008 - IEEE Standard for Floating-Point Arithmetic	2008	August 2008
About Netpbm (pfm)	http://netpbm.sourceforge.net/doc/pfm.html	April 2012
PKWARE .ZIP File Format Specification	6.3.3	September 2012

2 Definitions and Acronyms

2.1 Acronyms

Table 2-1: List of Acronyms

Acronym	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
MPAC	Multiple Projector Auto-Calibration (VESA Task Group)
MPCDI	Multiple Projector Common Display Interchange (VESA standard)
NaN	Not a number
PFM	Point Float Map
PNG	Portable Network Graphics
SW	Software
VESA	Video Electronics Standards Association
XML	Extended Markup Language

2.2 Glossary

Table 2-2: Glossary of Terms

Terminology	Definition
Alpha	A 0.0-1.0 ranged attenuation value that is multiplied to corresponding pixel data in order to modify its luminance, defined in Sections 3.9 and 3.11.
Beta	A 0.0-1.0 ranged offset value that is summed with corresponding pixel data in order to increase its luminance, defined in Sections 3.9 and 3.11.
Blending Box	Component of hardware which provides a smoothing of multiple images into the appearance of single continuous image
Frame Buffer (Buffer)	The entire area of a logical image. Typically the area of a bitmap
Image Generator	The 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, or a single graphics pipeline on a multi-head graphics card.
Projector	Device which projects light onto a surface to show images
Region	The area of a display (within the frame of reference of a frame buffer) covered or generated by an independent image rendering element.
Rendering Component	A 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.

Terminology	Definition
Software Based	Blending methodology which relies on algorithms defined in an application which is implemented in concert with a rendering component
Warping	The act of taking pixels in a particular spatial location and moving them to another spatial location.
Zip File	A compressed file which is packaged using algorithms which minimize the storage size of a data set

2.3 Rendering Component Definitions

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 depicted in Figure 2-1 in which three IGs are creating output that will drive a multi-input projector. In order 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.

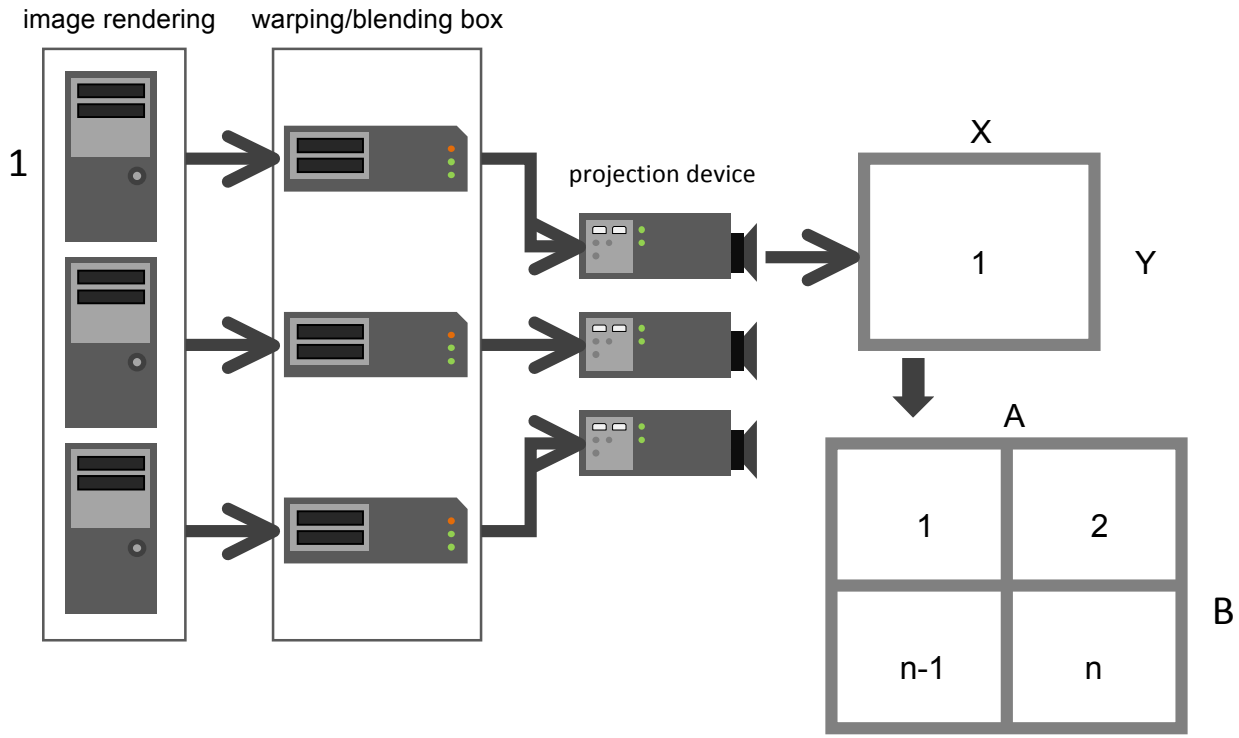

Figure 2-1: Elemental Relationships

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 which may be supported is one where 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 our 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.

2.3.1 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:

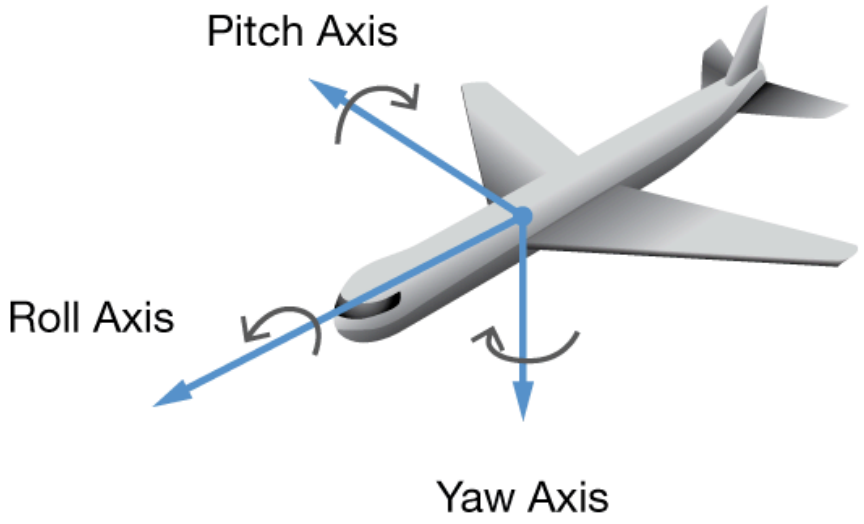

Figure 2-2: Representation of Rotations in the Viewing Matrix

2.3.1.1 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 in Table 2-3.

Table 2-3: Frustum View Matrix Definitions

Variable	XML	Definition	Units
Yaw	yaw	The viewer rotates his head to the right.	Degrees
Roll	roll	The viewer rotates his head clockwise.	Degrees
Pitch	pitch	The viewer rotates his head upwards.	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 upwards.	Degrees
Down Angle	downAngle	Field of View down. Negative angles mean see more downwards.	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(roll) × Pitch(pitch) × Yaw(yaw)
```

Where Roll(angle), Pitch(angle), and Yaw(angle) are functions that create matrices that perform a roll, pitch, and yaw rotation. The inverse of the *RotationMatrix* is:

```
RotationMatrix(From World to Viewer) = Yaw(-yaw) × Pitch(-pitch) × Roll(-roll)
```

To implement this rotation in standard method, the following code example is give as an aid to the user of this standard. This algorithm can only be applied to applications where OpenGL is used.

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 to the down. 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. *The code below is meant to be used as an example 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);
```

2.3.1.2 Coordinate Frame

Additional information is needed in addition to frustum definitions described in the previous section for Shader Lamp implementations. Since Shader Lamps require a full 3D geometry scan for every projector, the full 3D position and rotation of every projector must also be given. Thus, a coordinate frame along with a position must be given for each Shader Lamp projector. This can be expressed as three vectors and a position. Three vectors are necessary to resolve any left/right handed ambiguity.

Positions can simply be expressed as a 3D point. Each of the 3D vectors relates to the previous section of 2.3.1 in yaw, pitch, and roll. They will be expressed as orthonormal unit vectors for each of the yaw, pitch, and roll directions. Refer to Figure 2-2: Representation of Rotations in the Viewing Matrixfor a graphical representation of these unit vectors.

2.3.2 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 applied to it already. The gamma function is described as follows. Here, P_l is the linear value and the P_g is the gamma mapped value with γ gamma.

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

2.3.3 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 where P_i is the initial color, P_f is final pixel color, α is the gamma corrected intensity mapping color, and β is the gamma corrected black level offset color. Please note that all variables here are in the range of 0.0 to 1.0.

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

2.3.4 Lens Distortion Mapping

In the case of Shader Lamp implementations, precise pixel alignment is important. To achieve this, a lens distortion lookup table must be provided. This explicit lookup table 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 lookup tables can be stored exactly as 2D warps as the data is identical.

2.3.5 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. Since there are many methods of interpolation that can be used on any given dataset, the MPCDI supports the following interpolation schemes within its descriptors to describe how the geometry grid data was generated and in 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 grid of control points
- 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 describes the format of the interchange file.

3.1 Filename

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

[filename].mpci

3.2 File Format Description

The data file format shall utilize standard Extended Markup Language (XML) structures which contain specific containers and descriptors that provide for definition to determine the following elements of the blending data:

- Level of Integration Definition - XML descriptor file with optional frustum definitions – This will contain a profile number, a self-identifier, and a 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 - This 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) – A beta map may be used when a display requires a per-pixel intensity offset applied to the frame-buffer.

- Custom Definitions as determined - – In order 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.

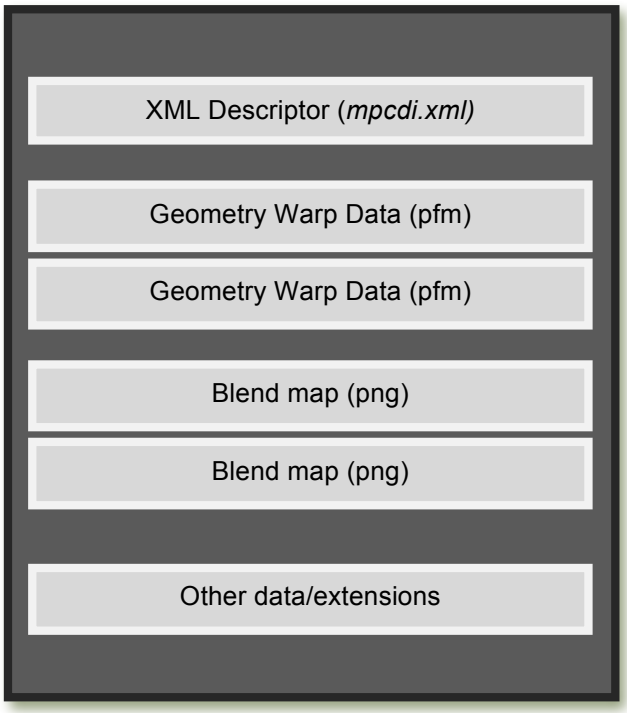

Figure 3-1: Container Format Construct

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 .mpci. Each of the individual components within the container will be individual files. These files are discussed in detail in further sections. 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 utilize 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 main use for the data, while the level dictates what the upper bound on fidelity can be for the given profile.

Profiles and levels are established to define the expectations of data structures for the varying levels in integration which are possible within a multiple projector display system. These profiles and levels are defined in Figure 3-2.

Profile Name	XML	Level 1	Level 2	Level 3	Level 4
2D Media	2d	1 or more projectors Warp grid not greater than 32x32 Single Channel alpha map support No Beta map support	All support of 1 Full RGB Beta map supported Full RGB alpha and beta map	All support of 1 Up to per pixel warp resolution	All support of 1,2, and 3
3D Simulation	3d	1 or more projectors Single Channel alpha map support Warp mesh no greater than every 10 pixels No beta map support	All support of 1 Full RGB Beta map supported Full RGB alpha and beta map	All support of 1 Up to per pixel warp resolution	All support of 1,2, and 3
Advanced 3D	a3	1 or more projectors 3d geometry of target surface 3d mesh no greater than every 10 pixels Single Channel alpha map support No beta map support	All support of 1 Full RGB Beta map supported Full RGB alpha and beta map	All support of 1 Up to per pixel 3d geometry	All support of 1,2, and 3
Shader Lamp	sl	1 or more projectors Projector location and rotation given 3d geometry of target surface Alpha map support No beta maps Up to per-pixel 3d geometry Lens distortions not supported.	All support of 1 Full RGB Beta map supported Full RGB alpha and beta map	All support of 1 Up to per-pixel lens distortion map supported	All support of 1,2, and 3

Figure 3-2: Profiles

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" which require definition for blending. This is pictorially shown in Figure 3-3.

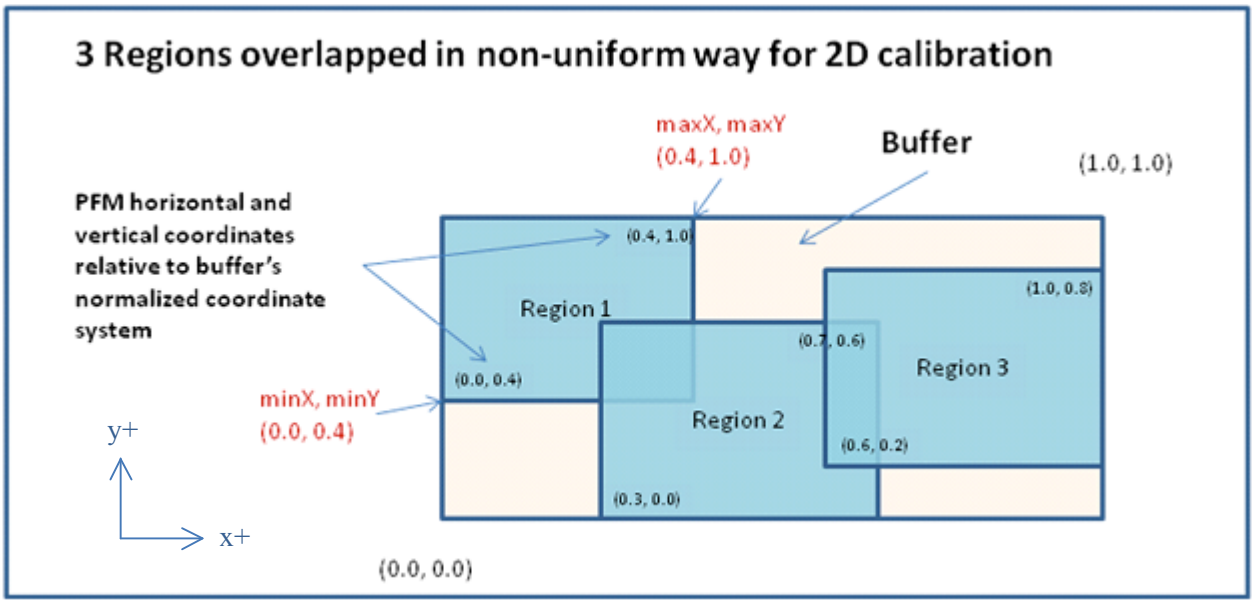

Figure 3-3: Buffer Map Diagram

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. This file shall be named *mpcdi.xml*. This 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 particularly important because we will be storing string values that equate to filenames.

- Directly following the XML version tag shall be the main *MPCDI* tag. This tag must contain the following attributes:

  ```
  <MPCDI profile="[profile]" level="[level]" date="[file generation time]"
  version="1.0">
  ```


- profile – Must be strings of *2D, 3D, a3,* or *sl*. See Figure 3-2
- level – Must be integer from 1 – 4. See Figure 3-2
- date – Stored as string in ISO 8601 format (yyyy-MM-dd HH:mm:ss)
- version – This must be set to the string "1.0" for this particular version of the MPCDI.

- For an *MPCDI* tag to be complete, it must contain one of each of the following sections denoted by the following tags:
  - *display* – Contains information about each display. See below for more detail.
  - *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 3.4.

- 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 *a3* or *sl*, there can only be **one** *region* tag within each *buffer* tag.
  - Each *buffer* tag must contain one or more *region* tag. Each *buffer* tag must contain an *id* attribute and optional *Xresolution* and *Yresolution* attributes. These are described below.
    - *buffer* tag attributes
      - *id* – This attribute contain a string specifying a unique identifier for this projection area. ID strings do not have to be consecutive numbers, but must be unique.
      - *Xresolution* – (Optional) An integer attribute specifying (in pixels) the horizontal resolution of the target display
      - *YResolution* – (Optional) An integer attribute specifying (in pixels) the vertical resolution of the target display
    - Each *region* tag shall contain the following attributes
      - *id* – Contains a unique string which identifies the area of projection being described
      - *XResolution* – contains an integer (in pixels) which defines the total horizontal resolution of the area being described
      - *YResolution* – contains an integer (in pixels) which defines the total vertical resolution of the area 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.

  - For cases where the profile is set to 2D Media (*2d*) and Advanced 3D Simulation (*a3*), the *region* tag shall have no tags contained within it.

  - 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 that describe the frustum as described in section 2.3.1.1. They are listed below and each of these tags shall each contain one floating point number stored in degrees.

      - *yaw*, *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 nine consecutive tags. They are as follows and is described in further detail in section 2.3.1.2.

    - Position: *posx, posy, posz*

    - Yaw vector: yaw*x, yawy, yawz*

    - Pitch vector: *pitchx, pitchy, pitchz*

    - Roll vector: *rollx, rolly, rollz*

- The *files* tag shall contain a list of all files needed for warping and blending. Each *files* tag shall contain one or more *fileset* tags and no other tags. Each *fileset* tag shall contain a *geometryWarpFile* tag and an *alphaMap* tag. *fileset* tags must contain a *betaMap* tag if levels 2 or 4 are specified in the *MPCDI* tag. If profile is set to Shader Lamp (*sl*), a *distortionMap* tag must be used.

  - *fileset* tag contains a region identifier which correlates to a buffer region which the files are correlated with. Within the *fileset* tag are the file identifiers.

  - 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.

    - *geometryWarpFile* tag must exist within the *fileset* tag

      - If the profile is *a3* or *sl*. Then a *geometricUnit* tag and an *originOf3DData* tag must both be present inside the *geometryWarpFile* tag.

- *geometricUnit* - states the unit of measurement the warp data is provided in. Valid values are the following strings: *mm, cm, dm, m, in, ft, yd, unknown*

- *originOf3DData* – States the origin of the 3d data. Valid values are the following strings: *centerOfMass, idealEyePoint, floorCenter, unknown*. This tag must be equal to all other *originOf3DData* tags that exist under the parent *file* tag. If the profile is set to Shader Lamps (*sl*), then this tag **cannot** be set to *idealEyePoint*.

- *interpolation* - states the preferred method for interpolating between control points of a warp mesh. Valid values are the following strings: *linear, keystone, smooth, and unknown*; as described in section 2.3.5

- An *alphaMap* tag must exist within the *fileset* tag with attributes described below.

- A *betaMap* tag must exist if profiles level is set to 2 or 4.

- Within the *alphaMap* and *betaMap* tag, there must be a *componentDepth* tag, a *bitDepth* tag. For *alphaMap* tags, an additional tag of *gammaEmbedded* tag must be included.

  - *componentDepth* – How many components does the alpha or beta map hold. Valid values are 1 or 3.

  - *bitDepth* – How many bits does a component have. Valid values are 8, 16, and 32.

  - *gammaEmbedded* – Stores a float inside the tag with the associated gamma value of the alpha map. Alpha with no gamma can use a value of 1.0 which would be equivalent to linear. The gamma equation is described in section 2.3.2.

- A *distortionMap* tag must exist within the *fileset* tag if profile is set to Shader Lamps (*sl*). Since distortion maps are always a fixed format, only a *path* tag shall exist under the *distortionMap* tag.

3.5.3 XML File

Please note that variables are denoted in square brackets [] and followed by the data type of the variable.

```
<?xml version="1.0" encoding="utf-8"?>
<MPCDI profile="[validProfileName*]" level="[validLevel*]" date="[file generation date]"
version="1.0">
  <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>
      </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*]</interpolationHint>
        </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>
       </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 two dimensional data or three dimensional data depending on which profile the current file supports. It will have following properties.

- Uniform grid – 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, but this problem should be minimized by file compression.

- Variable in size – Since warping data is a uniform grid, it can be stored at different resolutions. Different profile levels determine the maximum allowable size for the geometry warp grid. This allows for scalability to where 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.

- Fill layout of sub pixels – 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 while the bottom right of the last pixel will be the bottom right of the raster image.

- 32-bit floating point – Data will be stored as signed 32-bit floating numbers (IEEE 754).

- Little endian or big endian – This standard will not define endian-ness of the storage of floating point data. Endian-ness shall be stored properly according to the PFM format.

- Geometry Warp must be a minimum of a 2x2 grid. 1x1 mappings are not allowed.

3.6.1 Portable Float Map (PFM) Format

Geometry data will be stored as a three component (RGB) PFM file. PFM file is a very basic HDR image format. It offers 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 and the last pixel in the array is the bottom right pixel. Since we are using a uniform grid, geometry data maps well onto this format. This is an open format described at ***http://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. Two dimensional 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 fashion where the data stored maps source pixels coming from an IG or media server to the destination.

- These coordinates are normalized coordinates (0.0 – 1.0).

- The x coordinate and y coordinate of the mapping is stored into the R and G components of the pfm respectively.

- It should be noted that this type of lookup and storage is exactly the same as OpenGL UV mapping.

- 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 - 1.0. 0.0 denoting that there is no error where a measure of 1.0 denotes that the data is meaningless. This value is normalized on the output x and y coordinate. A value of 0.5 means 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).

- Places where 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 where the view frustum needs to update every frame. Three dimensional data will have the following properties:

- 3D data will be stored as world geometry co-ordinate space as a uniform grid of arbitrary size.

- X, Y, and Z will be stored in the R, G, and B components of the PFM file respectively.

- Since 3D data points maybe unknown, these values should be stored as NaN.

- Sub pixels should be interpreted the same way as 2D data.

- 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.

- The 3D can be following enumeration of real world units: *mm, cm, dm, m, in, ft, yard,* 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 origin of the data (0, 0, 0) shall be one of the following: center of mass, ideal eye point, floor center, 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 either one file, 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 format (png)

- Either file can be single or three components (Grey or RGB).
- Either file can be 8-bit or 16-bit per pixel
- Either file can be different in image size
- Files will be arbitrary in size
- Data can have a pre-applied gamma/power function applied. This will be specified in the XML file.
- Sub-pixels will be defined in the same manner as 2D geometry warp data.
- 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 a great way to store pixel data. It offers an open use as-is license. Furthermore it can offers lossless compression of any enumeration of image data.

3.8 Distortion Maps

Distortion maps are simply look up tables that map a 2D output image to another 2D output image. Distortion 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.

3.9 Custom File Definition

In order to make the file format more flexible, other files can be stored within the zip file so long as they are named and defined within the XML file.

3.9.1 XML Specification

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

3.9.1.1 Example 3D Color Lookup Table XML

Provided here is an example of what an extension may look like. This shows how a 3D color lookup table may be specified in the xml descriptor. This is merely **shown as an example** and is not part of the official standard.

```
<extension Name="3DColorLookup" hasExternalFiles="True">
```



```
    <defintions>
        <forwardLookUp id="fw0"></forwardLookUp>
        <inverseLookUp id="inv0"></inverseLookUp>
    </definitions>
    <fileSet>
      <fileseq id="fw0" numFiles="16">
        fw_00.png fw_01.png fw_02.png fw_03.png fw_04.png
        fw_05.png fw_06.png fw_07.png fw_08.png fw_09.png
        fw_10.png fw_11.png fw_12.png fw_13.png fw_14.png
        fw_15.png
      </fileseq>
      <fileseq id="inv0" numFiles="16">
        inv_00.png inv_01.png inv_02.png inv_03.png inv_04.png
        inv_05.png inv_06.png inv_07.png inv_08.png inv_09.png
        inv_10.png inv_11.png inv_12.png inv_13.png inv_14.png
        inv_15.png
      </fileseq>
    </fileSet>
</extension>
```


