GOOGLE

Dynamic Depth

Created: 2019-03-20
Version: 1.0

Contents

Overview

Preface

Augmented reality (AR) and depth photos are increasingly popular camera use cases across Android, however the OS lacks standardization in capturing, saving, and sharing such images.

A few independent solutions, primarily for depth photos, exist however none that extend across the ecosystem. By having uniformity and consistency within the Android camera framework, partners and developers can natively get support for Dynamic Depth with minimal effort.

Objective

Provide an easy to implement, quick to parse, and widely available depth extension to existing still imaging formats across Android and iOS.

The extension will support two primary cases, augmented reality and depth photos. By storing device-related metadata, this specification will extend common still imaging specs like JPEG, PNG, and HEIF while maintaining compatibility with existing image viewers.

Target audience

The primary audience for this document are engineers and product managers across:

A. device makers
B. image creation applications
C. image editing applications
D. image sharing applications
E. mobile chipset makers
F. mobile chipset solution providers

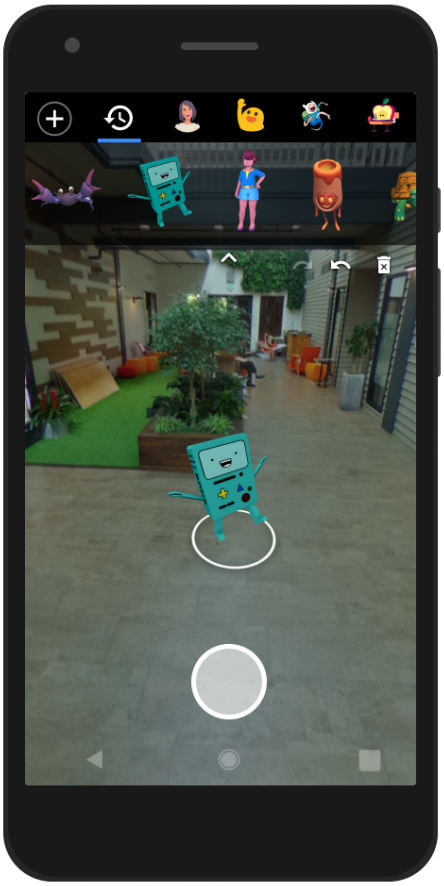
This is not an end-user facing specification nor does it contain end-user specific marketing material.

Use cases

Dynamic Depth is extensible to multiple types of depth use cases, optimizes for multi-camera sensors, and provides a foundation for computer vision/image processing extensions. The first two usages targeted are AR and depth photos.

Augmented Reality (AR) photos

In 2017 AR Stickers were introduced where a digital object (AR content) is placed into the scene currently viewed by the camera. An example of this is shown below with several objects being placed within a living room.



AR Sticker

In the scene above the AR content is placed while the viewfinder is active and capturing the scene. With Dynamic Depth a user would capture a scene, say the hallway shown above, and place AR content within the scene post-capture.

Depth photos

Depth photos can have various forms, some of the most popular ones include portrait mode found on many popular Android phones. An example of such an image is below:

Photo without (left) and with (right) depth of field effect.
Note how the effect's synthetic shallow depth of field helps suppress the cluttered background and focus attention on the main subject.

Today images like above cannot be edited once created. Starting with Dynamic Depth applications can seamlessly create bokeh effects post-capture.

These depth photos contain a depthmap which is defined as an image of values (integer or real) that represent distance from the view point (see example below). The exact definition of depth can vary depending on the depth sensor. As an example, two common definitions are depth along the optical axis (typically the z-axis), and depth along the optic ray passing through each pixel.

An example color image (left) and a corresponding depthmap (right).

Normative References

The following are normative references for this specification:

- Adobe XMP Specification Part 3 Storage in Files
- ISO 16684-1:2011(E) XMP Specification Part 1
- ISO/IEC 14496-12 ISO Box media file format
- T.81 (09/92) Digital Compression and Coding of Continuous-tone Still Images
- XML Schema Part 2: Datatypes Second Edition W3C Recommendation 28 October 2004

Data Structure

The metadata is serialized following ISO 16684-1:2011(E) XMP Specification Part 1 and embedded inside the primary image file as described in Adobe XMP Specification Part 3 Storage in Files.
The primary image file contains the following items, formatted as RDF/XML.

Elements

The root metadata object is the `Device` element and it is always required. This specification supports several Profiles or use cases and each has its own set of required elements and data. Readers may use Profiles to identify what use cases a given Dynamic Depth file can support. Multiple profiles can be supported by a single file. Readers and writers are not required to support all profiles.
The following table lists the minimum set of elements that are required for each Profile in this specification.

Profile	Required Elements

AR Photo	• Device
	  ○ Profile - must be “ARPhoto”
	  ○ Planes
	• Camera
	  ○ Pose
	  ○ LightEstimate
	  ○ ImagingModel
	  ○ Image (optional, see the AR Photo Profile for more info)
	  ○ DepthMap (optional)

Depth Photo	• Device
	  ○ Profile - must be “DepthPhoto”
	• Camera
	  ○ DepthMap
	  ○ Image

Optional elements are listed in the following list and may be ignored by image parsers that handle either of the Profiles above. Elements are defined in a separate section.

- Primary image - The image external to the Dynamic Depth, visible to normal non-Dynamic Depth apps
- `Device` - The root object of the RDF/XML document as in the Adobe XMP standard
    - `Container` - Ordered directory of concatenated files in the file container
    - `VendorInfo` - Vendor-related information for the device.
    - `AppInfo` - Application-specific or rendering information for the device.
    - `EarthPose` - The pose of the Realm (i.e. local world space) with respect to the earth.
    - `Pose` - The pose of the device with respect to the Realm.
    - `Profiles` - RDF sequence of one or more `Profile` entities
        - `Profile` - Defines the intended usage(s) of the Dynamic Depth metadata with the primary image.
    - `Cameras` - RDF sequence of one or more `Camera` entities
        - `Camera` - All the info for a given camera. There must be a camera for any image. The primary image is associated with the first camera, which is considered the primary camera for the image.
            - `VendorInfo` - Vendor-related information for the camera.
            - `AppInfo` - Application-specific or rendering information for the camera.
            - `Pose` - Camera pose relative to the Realm.
            - `Image` - Image provided by the camera
            - `ImagingModel` - Imaging (lens) model.
            - `DepthMap` - Depth-related information and the depth map.
            - `PointCloud` - Point-cloud data.
    - `Planes` - RDF sequence of one or more `Plane` entities
        - `Plane` - All the info for a given physical planar surface..

Media Data Encoding

Dynamic Depth files consist of a primary display-ready image, such as a JPEG file. Secondary images such as depth data, intermediate images, or alternative representations of the primary image, may be stored either in a concatenated file container defined below, or if the primary image is an ISO/IEC 14496-12 ISO Box media file format container, as other boxes in the container.

Concatenated File Container

The concatenated file container consists of a composite file where the primary image file has zero or more secondary media files appended to it. The secondary media files may contain alternative representations of the primary image or related media such as depth data.

The primary image contains a Container XMP metadata directory defining the order and properties of subsequent media files in the file container. Each file in the container has a corresponding media item in the directory. The media item describes the location in the file container and the basic

properties of each concatenated file. Media items in the container directory are referred to by ItemURI attributes from Image or Depth Map elements in the metadata.

ISO Box Media File Format Container

File containers based on ISO/IEC 14496-12 may be used to store the primary image, depth data, intermediate images, or alternative representations of the primary image. The XMP metadata in the container includes a Container XMP metadata directory element where each item in the directory uses a URI to refer to boxes within the ISO/IEC 14496-12 container.

Namespace Requirements

When Dynamic Depth is encoded in a JPEG container, all namespace declarations must appear in the main XMP section of the first 64K of the extended section. This allows clients to quickly create a list of the required namespaces by reading just those two sections (less than 128K), without having to load and parse the entire extended section.

Informative: If a Reader does not support all Dynamic Depth features for a particular application, it may be helpful for the reader to efficiently obtain a list of the Dynamic Depth namespaces (i.e., features and feature versions) used in a file before they begin processing it. Unfortunately, this can be difficult when using a JPEG container. If the `Device` element is more than 64K (true of most Dynamic Depth files), the rules of XMP force the `Device` and its children out of the main XMP section and into the extended section. Thus an Dynamic Depth element and its namespace declaration might appear anywhere in the main or extended XMP. Under these conditions, building a list of all the Dynamic Depth namespaces used in a file requires checking the entire Dynamic Depth content, often megabytes in length, causing a performance hit when opening the file.

Non-standard Usage

Writers are allowed to include other fields or objects that are not defined in the spec for the specified version of Dynamic Depth in `Device` elements. For example, these may be objects specific to a particular vendor, device, or use-case, or other extensions. Additions to the Device element must not change the behavior defined by the Dynamic Depth version value included in each element's namespace. Readers should ignore any additions that appear, without error.

Compatibility

The Dynamic Depth specification is a significant expansion of the original DepthMap Metadata specification published in 2014. It still supports the original use case of a single-image container with associated depth metadata, but expands that original specification to support more types of metadata and more use cases. The two specifications are not backwards compatible because depth media data is stored in a different way in this specification.

Readers and writers that supported the DepthMap Metadata spec will require modification to support Dynamic Depth. The Dynamic Depth standard handles a number of items differently, including: `Units`, `Confidence`, `Manufacturer`, `Model`, `ImageWidth`, and `ImageHeight`.

In this documentation, JPEG is used as the basic model, but the metadata definition may be applied to other file formats that support XMP.

Profiles

Profile elements describe the intended use of a Dynamic Depth image and define the structure of other required elements in the metadata. Profile elements allow Readers to quickly identify the intended use of a file. The `Profile` element contains the profile name and the indices of cameras used by the profile. Currently-supported use cases are depth photos and augmented reality (AR) photos.

The required Dynamic Depth elements for each profile are outlined below.

Profile and Use Case Extension

Future versions of the Dynamic Depth specification may add new Profiles to support new functionality and use cases. First, they can define the required Dynamic Depth elements, values, and semantics similar to the Depth Photo and AR Photo sections below. Next, their image reader and writer could support their use case name in the `Profile:Type` field, the list of Camera indices in `Profile:CameraIndices`.

Profile: Depth Photo

Depth photos are defined as an image of scalar values (integer or real) that represent the distance from the camera viewpoint, to the object viewed in a corresponding color image, see the figure below. The exact definition of the depth value may vary based on the type of depth sensor. As an example, two common definitions are depth along the optical axis (typically the z-axis), and depth along the optical ray passing through each pixel.

An example color image (left) and a corresponding depth image (right).

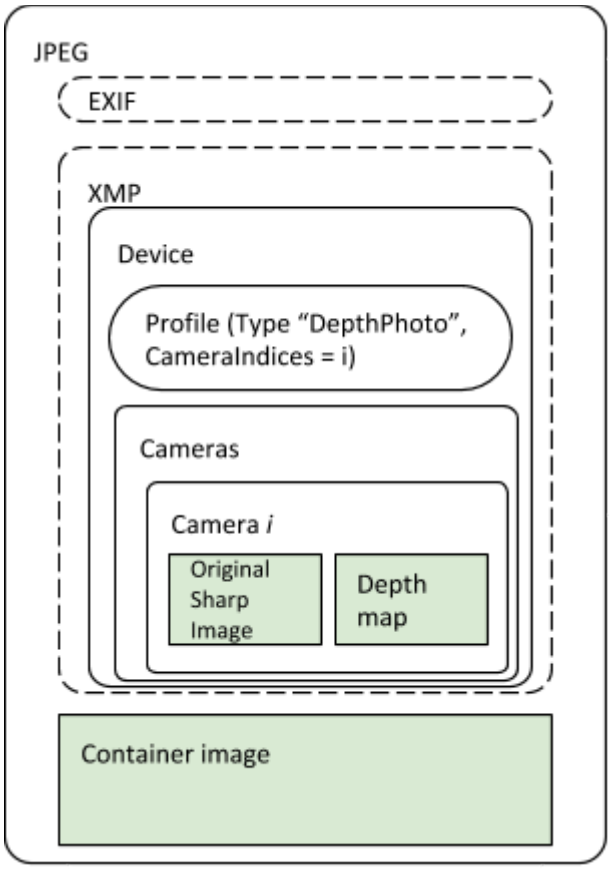
Required Depth Photo Elements

Profile

- `Profile:Type` must be set to `DepthPhoto`.
- `Profile:CameraIndices`. This list must have only one integer *i*, which represents the *i*th camera in the `Device:Cameras` list.

Cameras

- `Camera` *i*
  - `DepthMap` (1 or more)
  - `Image` (1 or more)



Dynamic Depth metadata structure for depth photography

Primary image

In depth photography, the primary image is the presentation or display-ready copy of the image. The image is not required if the camera index is 0 and the image and primary image are identical.

Image and depth map correlation

All images and depth maps within a `Camera` must be rectified to the same pose and cropped to the common field of view (same aspect ratio). It is not necessary for images and depth maps to have the same resolution.

Writers may store additional intermediate images inside subsequent Camera elements. Writers may store additional Camera elements for other intermediate representations of the color and depth images. Each additional Camera element may include a CameraPose element to define its position and orientation.

Profile: Augmented Reality (AR) Photo

An augmented reality (AR) photo is an image that contains the pose of the capturing device, lighting estimate information, horizontal and/or vertical surface planes in the world, and camera intrinsics.
Optionally, developers may include application-specific metadata. One example is the identifier(s) of 3D asset(s) and their poses. When the image is parsed, these assets can be loaded only by that application, with their poses, so that end-users may edit and move the asset interactively in the photo.

Optionally, a photo can be embedded inside an AR Photo. Should there be an embedded photo, it should be the one used by the image viewer when parsing back the photo. An example use case is yinwhere the embedded image is the one without 3D assets (which we will call the AR image), and the container has the 3D assets visible (the end-user facing image). This enables users to preview a thumbnail with the 3D assets in a gallery, while being able to interact and move objects in the AR image. Developers may choose to update the primary image if end-users save their edits to the AR photo.

If the embedded photo use case is employed, its parent Camera should be anything other than Camera 0, and it should hold all the AR metadata. Otherwise, if there is no embedded image, Camera 0 should contain all the AR metadata.

[Dynamic Depth Elements]

[Required]

Profile

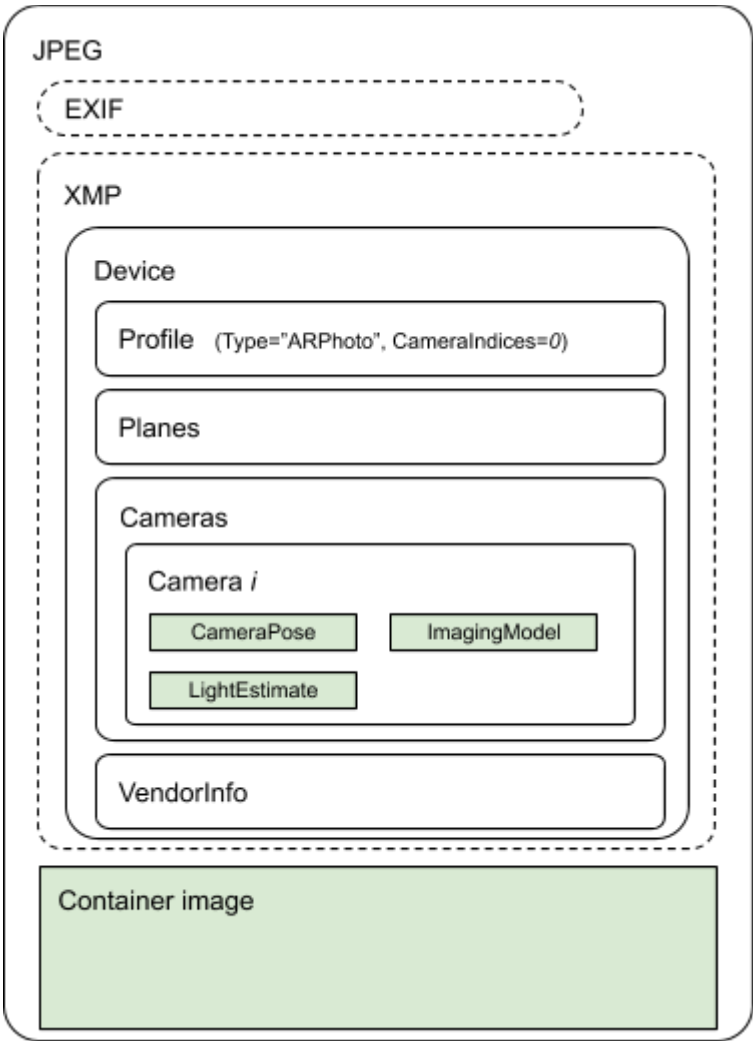
- `Profile:Type` must be set to `ARPhoto`.
- `Profile:CameraIndices`. This list must have exactly one integer *i*, which represents the *i*th camera in the `Device:Cameras` list.

Device

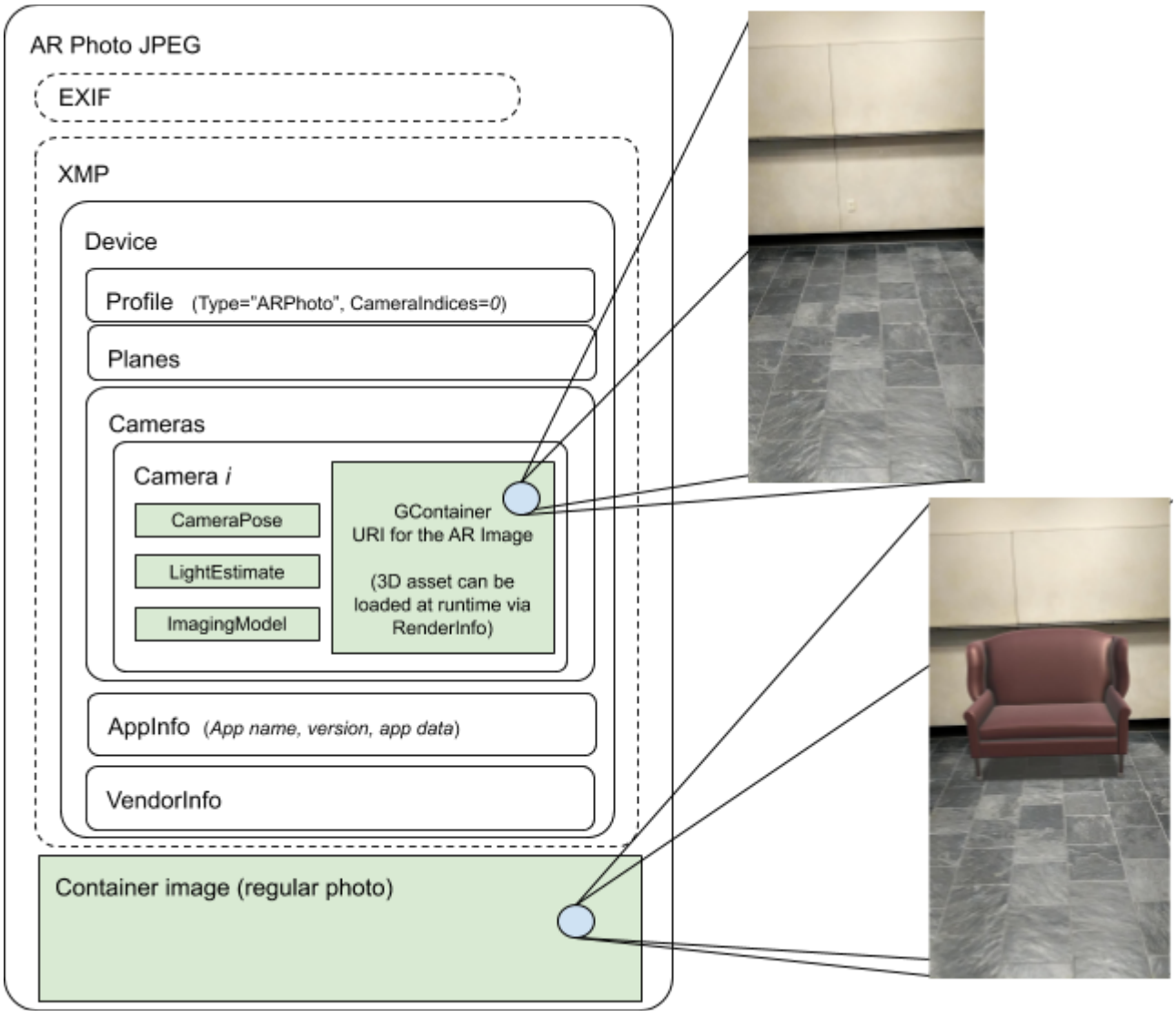
- `Planes` - a list of horizontal or vertical planes, representing detected physical surfaces in the world (e.g. floors or walls)

Camera

- `Camera` *i* - holds the AR metadata, and optionally the embedded AR image.
    - `Pose`
    - `LightEstimate`
    - `ImagingModel` - camera intrinsics information
    - `Image` - optional, AR image for the embedded use case



Dynamic Depth metadata structure for an AR Photo without an embedded image.



Dynamic Depth metadata structure for an AR Photo with an embedded AR image.

Poses and Coordinate Systems

Dynamic Depth stores the *pose* (that is, the *position* and *orientation)* of the camera(s) relative to the world, or the poses of objects (e.g. 3D assets) in the world. This enables applications to use multiple images together, as when mapping depth data onto a photograph, and provides information about the image capture, such as the position and orientation of the image sensor.

The following sections define the conventions, coordinate systems, and formats used throughout this specification. Most of the math involved can be handled by third-party math libraries. More details on additional coordinate systems, conventions, and math are available in the Appendix.

Conventions

Handedness. Unless otherwise noted, all Cartesian spaces are **right handed**. This means that cross(X, Y) == Z. Please see also the OpenGL section on handedness.

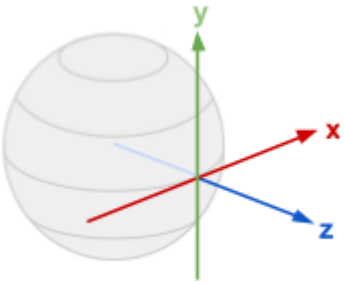
Position is expressed in three dimensions. For the device pose, these are latitude, longitude, and altitude. For the camera pose, they are the distance in meters from the device origin point along the device's x, y, and z axes.

Orientation is also expressed in three dimensions, as a rotation around x, y, and z axes relative to a frame of reference. For each Dynamic Depth element, the frame of reference is the local world coordinate system, which we define and describe below as the Realm coordinate system in this specification. For the Realm itself, the frame of reference is a standard "ENU" (east-north-up) earth coordinate system, described below.
Each of these 3D coordinate systems has a defined origin from which x, y, and z axes emerge in defined directions.

Dynamic Depth `Pose`. Dynamic Depth stores elements' Poses with respect to the local world coordinate system, which we define in this specification as the Realm coordinate system. The exception is EarthPose, which is the GPS reading of the Android device.
Note that some elements' pointwise locations are stored with respect to their local object coordinate system, such as with `Plane:Boundary`.
Below are the two major coordinate systems used in this specification. More details on the remainder are in the Appendix.

Earth Coordinate System

Dynamic Depth uses a right-handed, east-north-up (ENU) world coordinate system. This is the same world coordinate system used in the Android and iOS operating systems, and in ARCore.
The 3D position is represented in WGS84 coordinates as longitude, latitude, and altitude. In keeping with the WGS84 documentation, altitude is height in meters above the standard ellipsoid reference surface, and latitude is geodetic latitude. This is consistent with the GPS data provided by most mobile devices.



World coordinate system

Origin	The location specified by latitude, longitude, and altitude.
Orientation	• X is tangential to the ground at that location and points roughly East. (It is the vector cross product **y × z**.) • Y is tangential to the ground at that location and points towards the North Pole.

	• Z is perpendicular to the ground at that location and points towards the sky.
Units	Meters or degrees
Handedness	Right
Range	• Latitude: -90º to +90º • Longitude: -180º to +180º • Altitude: 0 to 100,000 meters (the edge of the atmosphere).
Precision	Double

Realm Coordinate System

The Realm coordinate system is an application's reference coordinate system in a real world. It is equivalent to ARCore's Session space, which serves as the reference space for all of the poses provided by its API.

Informational: The term "Realm" refers to this coordinate system, as to ensure clarity between this one and "Earth" space.

Origin	Arbitrary, depends on the application. • For rendering applications, see the origin of OpenGL's rendering world space. • For ARCore, it is generally the point in the real world when the user starts running the AR application.
Orientation	Local level with • Y up • Arbitrary X/Z axes, but generally -Z = projection of the device's "forward" vector at start-up time into the local-level plane. Note: "Forward" refers to the direction that a user is facing when they start the app with their device held out in front of them at arm's length, arm parallel to the ground.
Units	Meters
Handedness	Right
Range	Unbounded
Precision	Single

Object Coordinate System

This coordinate system is used for a Plane's points, and aligns with the Anchor coordinate system defined in ARCore. If applications choose to store serialized poses in `AppInfo`, it is recommended that they use this coordinate system as those objects' respective frames of reference.

The following definitions are recommended, but applications can choose to use the values that work best for their use case.

Origin	The center of the object on the XZ plane, and the bottom-most Y point on the object.
Orientation	+X right, +Y up, +Z out of the object, when looking head-on at the object.
Units	Meters
Handedness	Right-handed
Range	Depend on the use case
Precision	Single

Orientation data format

Mathematically, the task of describing orientation or rotation can be difficult and counterintuitive. Each of the popular formalisms for this – rotation matrix, Euler angles, axis-angle representation, and quaternions – has advantages and disadvantages. Dynamic Depth uses the quaternion representation, which is used throughout Android and ARCore APIs.
Informational: Readers can use library functions to convert it to matrices for calculations, if desired.

Transform Naming Conventions

Transformations are always named in the following form: <space_a>_T_<space_b> (with camel-casing for functions). For instance, this transform can map a point from the coordinate space_b into space_a coordinates. The "T" stands for "transformation" and should be pronounced "T" and not "to" to avoid confusion.

This naming makes it easy to see *visually* at a glance that the math is correct simply from the ordering of the space names. In the equation below the two "device" terms are adjacent to each other, and the order of "realm" and "camera" is preserved on both sides of the equation.
Recall that poses are just another name for rigid transformations. Since they are both abstract concepts and *not* implementations, we *do not distinguish between poses and matrices* where possible.

Example

- Camera poses are equivalent to `realm_T_camera`, which represents the pose of the camera in the Realm coordinate space.

- Coordinate system change. `Inverse` has the same semantics as an inverse matrix.

```
camera_T_realm = Inverse(realm_T_camera)

camera_T_plane = camera_T_realm * realm_T_plane;
```

- Linear point mapping: **`foo_T_bar`** `* bar_p`, where the *point bar_p* is transformed from frame bar to foo with a right sided vector multiplication (using homogeneous coordinates).

Element Definitions

This section defines and provides details of each Dynamic Depth element.

Device

The `Device` element contains information about the capture device and contains a sequence of Camera elements related to the primary image.

- The namespace URI is `http://ns.google.com/photos/dd/1.0/device`.
- The default namespace prefix is `Device`.

The first camera in the sequence is the one that created the primary image in the file container. At least one Camera element must be present in the Cameras element for a Device to be valid.

Name	Type	Required	Default Value	Property Content	If Container Image Modified
Profiles	Sequence of Profile (rdf:Seq)	No	N/A	Describes the intended purpose(s) of the image and its metadata. If the fields for more than one use case are present, all the applicable profiles should be listed.	No change
Cameras	Sequence of Camera (rdf:Seq)	Yes. Quantity depends on Profile	N/A	Each `Camera` in the `Cameras` sequence contains the properties of a camera on the device associated with this JPEG. If the first `Camera` in the sequence does not contain an `Image`, it references the primary image.	Needs update
Container	Container	Yes, if Image or DepthMap elements are present in any Camera element	N/A	Lists the directory of media items in all the Camera items under the Cameras element.	No change
Planes	Sequence of Plane (rdf:Seq)	Depends on Profile	N/A	Each `Plane` in the `Planes` sequence contains the properties of a vertical or horizontal plane in the world, such as a wall or floor surface.	No change
EarthPose	EarthPose	No		The pose of the Realm, with respect to the Earth. Please see the description in the Poses and Coordinate Systems section.	No change
Pose	Pose	No		The pose of the `Device`, with respect to the Realm. Please see the description in the Poses and Coordinate Systems section.	
VendorInfo	VendorInfo	No		Vendor information for the device	No change
AppInfo	AppInfo	No		Application information for this device	

Container Element

The container element is encoded into the XMP metadata of the primary image and defines the directory of media items in the container. Media items must be located in the container file in the same order as the media item elements in the directory and must be tightly packed.

- The namespace URI is `http://ns.google.com/photos/dd/1.0/container`.
- The default namespace prefix is `Container`.

The directory may contain only one primary image item and it must be the first item in the directory.

Element Name	Type	Description
Directory	Ordered Array of Structures	Ordered array of Container:Item structures defining the layout and contents of the container.

Item Element

Media item elements describe how each item should be used by the application.

The first media item must be the primary image. The primary image is usually the container image, in which case the required field is the MIME type. The Length field must be zero for this case.

In general, an Item must contain a Mime attribute specifying one of the image MIME types listed in Item MIME Type Values. The length of the primary item may be determined by parsing the primary image based on its MIME type starting at the beginning of the file container.

The first media item may contain an Padding attribute specifying additional padding between the end of encoded primary image and the beginning of the first secondary image. Subsequent media items for secondary images may not contain Padding attributes.

Each media item must contain an Mime attribute. The secondary media items must also contain Length attributes.

Sequential media items may share resource data within the file container. The first media item determines the location of the resource in the file container, and subsequent shared media items have Length set to 0. In the case that the resource data is itself a container, DataURI may be used to determine the location of the media item data within the resource.

The location of media item resources in the container is determined by summing the Length's of the preceding secondary item resources to the length of the primary image encoding plus Padding if specified.

Attribute Name	Type	Description
Mime	String	Required. Simple string indicating the MIME type of the media item in the container.
Length	Integer	Required for secondary media items. Simple string containing a positive integer length in bytes of the item. Media items are expected to be in their original form, with no encoding applied. The length value is the actual length of the bytes in the file. Length 0 in secondary media items indicates that the media item resource is shared with the previous media item. Length is expected to be 0 in a primary media item.
Padding	Integer	Optional for the primary media item. Simple string containing a positive integer length in bytes of additional padding between the end of encoded primary image and the beginning of the first secondary image.
DataURI	String	Required if there is an element that references this `Container`, such as `Image`, `Depth Map`, or `AppInfo`. Applications should choose a URI that makes it easy for readers to identify that the associated item belongs to the application. Directory paths are a good example of a DataURI's ideal structure. Otherwise, optional for ISO base media format ISO/IEC 14496-12 mime types. URI string conforming to ISO/IEC 14496-12 8.11.9 containing the relative URI of the media data inside the media item resource.

Item Mime Type Values

The ItemMime attribute defines the MIME type of each media item.

Value	Description
image/jpeg	JPEG Image
image/png	PNG Image
image/tiff	TIFF compatible image
image/heif, image/heic	HEIF image
text/plain	Text data

Profile

The `Profile` element describes the use case of the photo being captured.

- The namespace URI is `http://ns.google.com/photos/dd/1.0/profile`.
- The default namespace prefix is `Profile`.

Name	Type	Required	Default Value	Property Content	If Container Image Modified
Type	string	Yes		DepthPhoto, ARPhoto	No change
CameraIndices	Sequence of integers (rdf:Seq)	Depends on the use case as defined in the Profile	N/A	Indicates the cameras that will be used in the profile. See the respective profile description for the intended use of each camera. Camera indices here are independent of indices defined or mandated by Android camera framework.	No change

Vendor Information

The `VendorInfo` element describes vendor information for a camera or a device.

- The namespace URI is `http://ns.google.com/photos/dd/1.0/vendorinfo`.
- The default namespace prefix is `VendorInfo`.

Name	Type	Required	Default Value	Property Content	If Container Image Modified
Model	string	No		The model of the element that created the content	No change
Manufacturer	string	Yes	N/A	The manufacturer of the element that created the content	No change
Notes	string	No		General comments	No change

Application Information

The `AppInfo` element describes application-specific information for the given image. Example use cases include a projection mesh of the primary image, or 3D assets' identifiers and their poses. Possible ways of serializing this data include a little-endian base64-encoded string, or using JavaScript Object Notation. This data would then be stored in a Container with a text mime type.

The flexibility of custom serialization mechanisms aim to enable experimentation and/or maintaining per-application use of this data, to name a few use cases. Applications are expected to define their own mechanisms for parsing this data as well.

- The namespace URI is `http://ns.google.com/photos/dd/1.0/appinfo`.
- The default namespace prefix is `AppInfo`.

Name	Type	Required	Default Value	Property Content	If Container Image Modified
Application	string	Yes	N/A	The model of the element that created the content	No change
Version	string	Yes. Otherwise the ItemURI value must be present.	N/A	The version of this application's data serialization scheme.	No change
ItemURI	string	Yes. Otherwise Version must be present		The URI of the Container that holds the custom data used by this application. Image readers who do not recognize the application name should not attempt to read the associated Container.	No change

Earth Pose

The `EarthPose` element describes a *pose* (i.e., *position* and *orientation*) with respect to the Earth coordinate system. In Dynamic Depth, this is used for describing the pose of the Realm with respect to the Earth in Device:RealmPose. Please see the Poses and Coordinate Systems section for details on how the Earth, Realm, Device, Camera, and Entity coordinate systems work together.

- The namespace URI is `http://ns.google.com/photos/dd/1.0/earthpose`.
- The default namespace prefix is `EarthPose`.

The raw data used to determine the Realm's pose (with respect to the Earth) may come from GPS and IMU sensors. In most cases, this is equivalent to the GPS reading. When providing the position, all three of the position fields are required, and when providing the orientation, all four orientation fields are required.

Position data shows the Realm's location on the Earth. **Rotation (orientation) data** shows the containing entity's orientation relative to the ENU world coordinate system, in the quaternion format as described under Poses and Coordinate Systems.

Name	Type	Required	Property Content	If Container Image Modified
Latitude	double	Yes, if providing position.	WGS84 latitude in degrees	No change
Longitude	double	Yes, if providing position.	WGS84 longitude in degrees	No change
Altitude	double	Yes, if providing position.	WGS84 altitude in meters	No change
RotationX	real	Yes, if providing orientation.	The x component of the quaternion representation.	No change
RotationY	real	Yes, if providing orientation.	The y component of the quaternion representation.	No change
RotationZ	real	Yes, if providing orientation.	The z component of the quaternion representation.	No change
RotationW	real	Yes, if providing orientation.	The w component of the quaternion representation.	No change
Timestamp	long	Depends on use case	Time of capture, in milliseconds since the Epoch (January 1 1970, 00:00:00.000 UTC).	No change

Pose

The `Pose` element describes the *pose* (i.e., *position* and *orientation*) of its container element with respect to the Realm. Please see the Poses and Coordinate Systems section for details on how the Earth, Realm, Device, Camera, and Entity coordinate systems work together.

- The namespace URI is `http://ns.google.com/photos/dd/1.0/pose`.
- The default namespace prefix is `Pose`.

The position and orientation of the device, each camera, and each plane relative to the Realm are based on information provided by the manufacturer, Android APIs, or AR APIs. Image-creation apps need this information in order to create the Dynamic Depth file. Image consumers do not need this information; they just need the pose data.

If it is not possible to know the device pose relative to the Realm, it is assumed to have the same pose as that of Camera 0, which is the primary camera. Both are assumed to have the same pose, and the `DevicePose` is set to identity (no difference).

For non-AR use cases, the pose of the device (with respect to the Realm) is the zero pose (i.e. identity matrix), and this field will not be needed.

Pose is also used in an Entity context when describing the pose of a Plane. Applications may also choose to adopt this usage convention for customized use cases, such as encoding 3D assets and their poses in the AppInfo Element.

Position data shows the x, y, z coordinates of the center of the device, plane, or camera lens with respect to the Realm, in meters where possible. **Rotation (orientation) data** shows the device, plane, or camera orientation relative to the Realm, in quaternion format as described under Poses and Coordinate Systems.

The table below shows the components of the `Pose` element.

Name	Type	Required	Default Value	Property Content	If Container Image Modified
PositionX	real	Yes, if providing position.	0	The x position in meters, relative to the Realm.	No change
PositionY	real	Yes, if providing position.	0	The y position in meters, relative to the Realm.	No change
PositionZ	real	Yes, if providing position.	0	The z position in meters, relative to the Realm	No change

RotationX	real	Yes, if providing orientation.	0	The x component of the quaternion representation.	No change
RotationY	real	Yes, if providing orientation.	0	The y component of the quaternion representation.	No change
RotationZ	real	Yes, if providing orientation.	0	The z component of the quaternion representation.	No change
RotationW	real	Yes, if providing orientation.	0	The w component of the quaternion representation.	No change
Timestamp	long	Yes		Time of capture, in milliseconds since the Epoch (January 1 1970, 00:00:00.000 UTC).	No change

Camera

The `Camera` element describes a camera imaging sensor, and must reference an image. This image may be a Container URI, or the primary image if the given camera is the first one in the `Cameras` list (i.e. `Camera 0`).

- The namespace URI is `http://ns.google.com/photos/dd/1.0/camera`.
- The default namespace prefix is `Camera`.

The first `Camera` that appears in the data structure is referred to in this documentation as Camera 0. In a simple case (e.g., a smartphone or tablet), Camera 0 should be the primary camera, and may reference the primary image; the rest of the Camera sequence is arbitrary. The semantics of each camera are defined by the Profiles and specified use cases; please see the Profiles section for a detailed description.

All images and depth maps within a single `Camera` element are presumed to be rectified that same Camera. That is, the elements have the same pose, proportions, and field of view. Any additional `Camera` element should either include accurate pose data for that camera relative to the Realm.

Name	Type	Required	Default Value	Property Content	If Container Image Modified
Trait	string	Depends on Profile	“Physical”	One of “Logical” or “Physical” to indicate the nature of this camera. Assumed Physical if this field is not present.	No change
DepthMap	DepthMap	Depends on Profile		The `DepthMap` property of this `Camera`	If image scaled or cropped, update accordingly
Image	Image	Yes		The `Image` property of this `Camera`	If image scaled or cropped, update accordingly
PointCloud	PointCloud	Depends on Profile		The `PointCloud` property of this `Camera`	If image scaled or cropped, update accordingly
ImagingModel	ImagingModel	Depends on Profile		The imaging model of this `Camera`.	If image scaled or cropped, update accordingly

Pose	Pose	Depends on Profile		The pose of this `Camera`. The poses of `Camera` elements are all relative to the Realm.	No Change
LightEstimate	LightEstimate	Depends on Profile		The lighting estimation data of this `Camera`.	No change
VendorInfo	VendorInfo	Depends on Profile		Vendor info for this camera	No change
AppInfo	AppInfo	Depends on Profile		Application-specific data for this camera	Depends on use case

Image

The `Image` element contains a reference to a color image stored elsewhere in the file container.

- The namespace URI is `http://ns.google.com/photos/dd/1.0/image`.
- The default namespace prefix is `Image`.

Name	Type	Required	Default Value	Property Content	If Container Image Modified
ItemSemantic	string	Yes.	N/A	A defined ItemSemantic value, which describes the intended use or contents of the image. See table below.	No change.
ItemURI	string	Yes	N/A	The URI of the Container for this camera's image.	No change.

Informative: One use of the Image element is described under Profile: Depth Photo, in which a backup of the original image is stored along with the matching depth data in the Camera 0 Image, and the primary image is treated as a "presentation" or "display" copy.

For other uses, such as storing an infrared photograph to accompany a normal color photograph, it's better to put the `Image` in a separate `Camera`. It's likely that this approach will correspond to the actual equipment – for instance, an infrared image is taken by a separate IR camera on the same device. The additional `Camera` element should either include accurate pose data for that camera relative to the device, or have no pose data, indicating that the image has already been rectified to the Camera 0 image.

ItemSemantic

The `Image:ItemSemantic` attribute defines the intended use of each image captured by a Camera. In addition to the Depth semantic defining the container element storing the depth map data, the following item semantics may be used:

Value	Description
Primary	Indicates that the item is the primary display ready image in the container. The container may have only one Primary item.
Original	Indicates that the media item is an original version of the primary image, such as an unfiltered color image that may be used to render a depth effect.

Light Estimate

The `LightEstimate` element provides the color correction RGB values and average intensity of a real-world scene, as seen by an image sensor.

- The namespace URI is `http://ns.google.com/photos/dd/1.0/lightestimate`.
- The default namespace prefix is `LightEstimate`.

Informative: This data can be obtained using the ARCore APIs. These values are conventionally used for adjusting the color of virtual objects when they are rendered in an image, such as in OpenGL fragment shaders.

Name	Type	Required	Default Value	Property Content	If Container Image Modified
ColorCorrectionR	float	Yes, if G or B are present. Optional otherwise.	1.0	The red color correction scaling factor to be applied to the final color computed by the fragment shader to match the ambient color.	Scale/crop: No change
ColorCorrectionG	float	Yes, if R or B are present. Optional otherwise.	1.0	The green color correction scaling factor to be applied to the final color computed by the fragment shader to match the ambient color.	Scale/crop: No change
ColorCorrectionB	float	Yes, if R or G are present. Optional otherwise.	1.0	The blue color correction scaling factor to be applied to the final color computed by the fragment shader to match the ambient color.	Scale/crop: No change
PixelIntensity	float	Yes	1.0	The average pixel intensity of the image sensor.	No change.

Plane

The `Plane` element describes a vertical or horizontal planar surface in the world.

- The namespace URI is `http://ns.google.com/photos/dd/1.0/plane`.
- The default namespace prefix is `Plane`.

Informative: This data can be obtained using the ARCore APIs.

Name	Type	Required	Default Value	Property Content	If Container Image Modified
Pose	Pose	Yes	N/A	The pose of this `Plane`.	No Change
ExtentX	float	Optional	-1	The extent of the plane in the X dimension, centered on the plane position.	No change
ExtentZ	float	Optional	-1	The extent of the plane in the Z dimension, centered on the plane position.	No change
BoundaryVertexCount	Integer (even)	Yes, if providing Boundary	0	The number of vertices in the enclosing boundary polygon.	No change
Boundary	string	Yes, if providing BoundaryVertexCount	N/A	A little-endian base64-encoded list of (float) vertex pairs, representing the enclosing polygon vertices on the XZ plane. These points are in the Plane's local coordinate system. Please see the definition of the Object Coordinate System in this specification.	No change

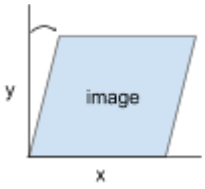
Imaging Model

The `ImagingModel` element describes the imaging model of a camera lens.

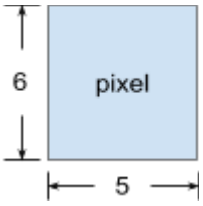
- The namespace URI is `http://ns.google.com/photos/dd/1.0/imagingmodel`.
- The default namespace prefix is `ImagingModel`.

The imaging model can be used to describe a fisheye distortion model, or a standard pinhole camera model with 5-DoF radial distortion.

Skew is the clockwise angle by which the y axis of the image slants away from the vertical.



The **pixel aspect ratio** is the x/y ratio of pixel width to pixel height. If pixels are perfectly square, the ratio is 1/1 = 1.0 (the default). In the image below, the ratio would be 5/6 = 0.83. When the pixel aspect ratio is 1.0 (the default), this `ImagingModel` element describes a standard pinhole camera model with 5-DoF radial distortion. In this model, *skew* is the clockwise angle by which the y axis of the image slants away from the vertical.



The lens distortion model supports the Brown-Conrady model. The `Distortion` field stores a list of value pairs [k_1, p_1, k_2, p_2, …, k_n, p_n] and supports a variety of parameter configurations:

- For applications using LENS_RADIAL_DISTORTION, kappa_0 through kappa_3 maps to k_1 through k_4, while kappa_4 to kappa_5 maps to p_1 and p_2. Note that p_3 and p_4 must be zero in this case, so the stored values in `Distortion` are:

  [k_1, p_1, k_2, p_2, k_3, p_3, k_4, p_4] = [kappa_0, kappa_4, kappa_1, kappa_5, kappa_2, 0, kappa_3, 0]

  where the lens distortion correction equations are:

```
x_c = x_i * (kappa_0 + kappa_1 * r^2 + kappa_2 * r^4 + kappa_3 * r^6)
      + kappa_4 * (2 * x_i * y_i) + kappa_5 * (r^2 + 2 * x_i^2)
y_c = y_i * (kappa_0 + kappa_1 * r^2 + kappa_2 * r^4 + kappa_3 * r^6)
      + kappa_5 * (2 * x_i * y_i) + kappa_4 * ( r^2 + 2 * y_i^2 )
```

- For applications using LENS_DISTORTION, kappa_1 through kappa_3 maps to k_1 through k_3, while kappa_4 to kappa_5 maps to p_1 and p_2. Note that p_3 must be zero in this case, so the stored values in `Distortion` are:

  $[k_1, p_1, k_2, p_2, k_3, p_3]$ = [1, kappa_4, kappa_1, kappa_5, kappa_2, 0, kappa_3, 0]

  where the lens distortion correction equations are:

```
x_c = x_i * (1 + kappa_1 * r^2 + kappa_2 * r^4 + kappa_3 * r^6) +
      kappa_4 * (2 * x_i * y_i) + kappa_5 * ( r^2 + 2 * x_i^2 )
y_c = y_i * (1 + kappa_1 * r^2 + kappa_2 * r^4 + kappa_3 * r^6) +
      kappa_5 * (2 * x_i * y_i) + kappa_4 * ( r^2 + 2 * y_i^2 )
```

- For other parameter configurations of the Brown-Conrady model, such as the 2-polynomial [k1, k2] or 3-polynomial [k1, k2, k3], zeroes must be the value for any p_i parameter that is not used.

Name	Type	Required	Default Value	Property Content	If Container Image Modified
FocalLengthX	real	Yes	N/A	The focal length of the lens along the X axis, normalized by the maximum dimension of the sensor. I.e., given the focal length in pixels f_x and the size of the sensor in pixels (*width*, *height*), then: $FocalLengthX = f_x \,/\, max(width, height)$	If image cropped, update accordingly
FocalLengthY	real	Yes	N/A	The focal length of the lens along the Y axis, normalized by the maximum dimension of the sensor. I.e., given the focal length in pixels f_y and the size of the sensor in pixels (*width*, *height*), then: $FocalLengthY = f_y \,/\, max(width, height)$	If image resized or cropped, update accordingly
PrincipalPointX	real	No	0.5	The x position indicating where the camera optical axis crosses the image plane center of the camera along the X axis, normalized by the sensor width.	If image resized or cropped, update accordingly
PrincipalPointY	real	No	0.5	The y position indicating where the camera optical axis crosses the image plane center of the camera along the Y axis, normalized by the sensor height.	If image resized or cropped, update accordingly
ImageWidth	integer	Yes, if ImageHeight is present	N/A	The width of the image, in pixels.	If image resized or cropped, update accordingly

ImageHeight	integer	Yes, if ImageWidth is present	N/A	The height of the image, in pixels.	If image resized or cropped, update accordingly
Skew	real	No	0	The skew of the image camera in degrees	
PixelAspectRatio	real	No	1.0	The aspect ratio of the X scale factor over the Y scale factor (defined above).	
DistortionCount	integer	Yes, if Distortion is present	0	The number of distortion parameter pairs in the Distortion field. That is, the total number of values in Distortion is twice this number.	
Distortion	string	Yes	N/A	Little-endian base64 serialization of a list of Brown-Conrady distortion parameters $[k_1, p_1, k_2, p_2, \ldots, k_n, p_n]$ where: $k_1, k_2, \ldots, k_n$ are radial distortion coefficients, and $p_1, p_2, \ldots, p_n$ are tangential distortion coefficients. Please see the description above on appropriate storage values.	

Depth Map

The `DepthMap` element contains a depth map image and information about its creation and format.

- The namespace URI is `http://ns.google.com/photos/dd/1.0/depthmap`.
- The default namespace prefix is `DepthMap`.

Depth maps are images of integer or real values that represent distance from the view point to a viewed object. The definition of depth values can vary depending on the type of depth sensor. For example two common definitions are depth along the optical axis (typically the Z axis), and depth along the optical ray passing through each pixel. That is, the distance of an object from the plane perpendicular to the Z axis, versus the distance from the object directly to the camera lens. The `MeasureType` element specifies which definition is used.

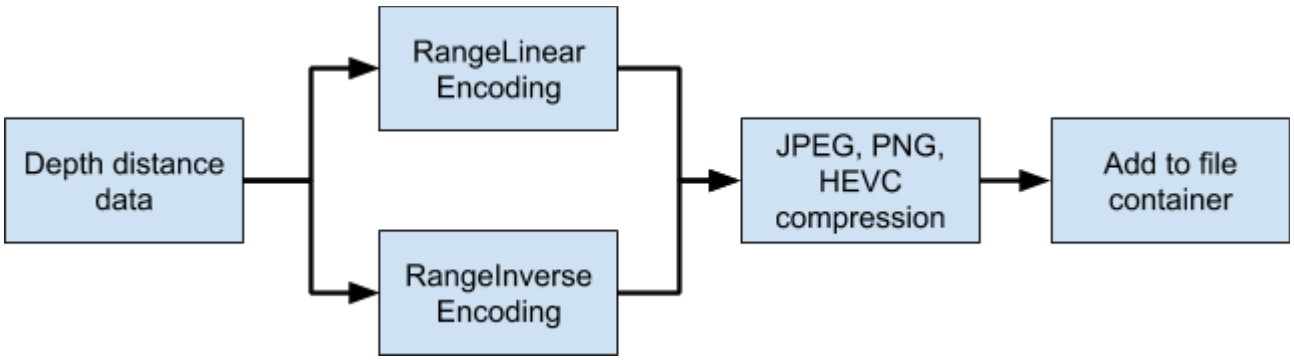
Writers may store depth maps using only Camera 0, or may rectify images from additional cameras before storing them (i.e., adjusting the depth data as if it had been captured at the pose of Camera 0, and cropping both images down to just the overlapping area). Writers may store the `DepthMap` under Camera *i* (as defined in the respective `Profile`) along with the primary image.

Depth images for the first camera must have no holes. The writer must encode an estimated value in any region where depth value cannot be calculated.

Depth Data

The depth map distance data is serialized into an image format and then stored as a separate item in the file container. The encoding pipeline contains two steps:

1. Convert from the input format (e.g., float or int32 values) to an integer grayscale image format as 16-bit words.
2. Compress using an image codec supported by the file container type.



Informative: The pipeline can be lossless or lossy, depending on the number of bits of the original depth map.

Two conversion formats are supported: **RangeLinear** and **RangeInverse**. **RangeInverse** is the recommended format if the depth map will lose precision when encoded, such as when converting from float to 16-bit. RangeInverse allocates more bits to the near depth values and fewer bits to the far values.

RangeLinear

Let d be the depth distance value of a pixel, and near and far the minimum and maximum depth values considered. The depth value is first normalized to the [0, 1] range as:

$$d_n = \frac{d - near}{far - near}$$

then quantize to 16 bits as:

$$d_{16bit} = \lfloor d_n \cdot 65535 \rfloor$$

Conversely, given the quantized depth d_n, one can recover depth d as:

$$d = d_n \cdot (far - near) + near$$

RangeInverse

Let d be the depth distance value of a pixel, and near and far the minimum and maximum depth values considered. The depth value is first normalized to the [0, 1] range as:

$$d_n = \frac{far \cdot (d - near)}{d \cdot (far - near)}$$

then quantize to 16 bits as:

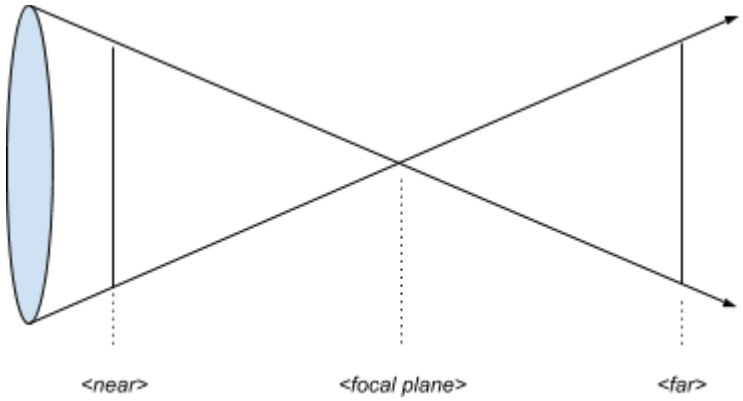
$$d_{16bit} = \lfloor d_n \cdot 65535 \rfloor$$

Conversely, given the normalized depth d_n, one can recover depth d as:

$$d = \frac{far \cdot near}{far - d_n \cdot (far - near)}$$

FocalTable

Writers may optionally include metadata describing a lens focal model for the depth data. Readers may use this to render a depth-of-field effect. The model defines the radius of the circle of confusion at different distances from the viewer. Distance and radius pairs are used to construct a lookup table defining the circle of confusion as a function of depth distance value.



The FocalTable attribute is a string value consisting of base-64 encoded little endian floating point pairs, which are actual distance values, not 8 or 16 bit quantized and encoded values. These *<distance>, <radius>* pairs define a lookup table that may be used to compute the radius of the circle of confusion at distance values between the near and far values of the depth map. Pairs must appear in ascending order sorted by the distance value. Distance coordinates are defined in depth map distance value Units. Radius values are defined in pixel coordinates. Radius values must be greater or equal to zero. The radius value zero denotes an in-focus depth distance on the focal plane of the image.

The lookup table must contain at least two pairs for the near and far values of the depth map. Radius values are linearly interpolated between points defined in the table.

Informative: Most applications require lookup table with three values for the near plane, focal plane of the image, and the far plane. Objects at the focal plane depth value would be rendered in-focus. Readers should accept focal tables with more than one distance value in focus.

Confidence Maps

The confidence values can be interpreted either directly as [0, 255] after de-compression or the client should normalize to [0.0, 1.0f], where 0 means no confidence, 1.0 means 100% confidence, and everything in between is a linear interpolation between 0% and 100%. These values have the same semantics as the confidence definition in Android's DEPTH16 format.

Confidence maps are not range-encoded or compressed, and near/far values are not to be used in the maps.

ItemSemantic

The `DepthMap:ItemSemantic` attribute defines the intended use of each depth map captured by a Camera.The following item semantics may be used:

Value	Description
Depth	This field's default value. Indicates that the depth image is intended to be used as a depth map.
Segmentation	Indicates that the depth image is intended to be used as a segmentation mask.

Depth Map Definition

Name	Type	Required	Default Value	Property Content	If Container Image Modified
Format	string	Yes	N/A	The conversion format used to encode depth: "RangeInverse" or "RangeLinear"	Scale/crop: No change
ItemSemantic	string	Yes	Depth	String value defined above describing the intended use or contents of the depth image. See table above.	No change.
Near	real	Yes	N/A	The near distance value of the depth map. If "Units" is set to "Meters", the units are meters. Otherwise the units are undefined.	Scale/crop: No change
Far	real	Yes	N/A	The far distance value of the depth map. If "Units" is set to "Meters", the units are meters. Otherwise the units are undefined.	Scale/crop: No change
Units	string	Yes	None	The units of the depthmap, i.e. "Meters" for meters, "Diopters" for non-metrically accurate data, or "None" for no units.	No change
DepthURI	string	Yes	N/A	The URI of the Container for the depth image.	Scale: No change as long as aspect ratios match. Crop: Decode data into an image, crop to matching ratio, then re-encode.

ConfidenceURI	string	No	N/A	The URI of the Container for the confidence map. The container item must support 16 bit data.	Scale: No change as long as aspect ratios match. Crop: Decode data into an image, crop to matching ratio, then re-encode.
MeasureType	string	No	"OpticalAxis"	The type of depth measurement. Current valid values are "OpticalAxis" and "OpticRay". "OpticalAxis" measures depth along the optical axis of the camera, i.e., the Z axis. "OpticRay" measures depth along the optic ray of a given pixel.	No change
Software	string	No	N/A	The software that created this depth map	No change
FocalTableEntryCount	integer	Yes if FocalTable is present	N/A	The number of pairs (i.e. entries) in FocalTable. This value must be no less than 2.	No change
FocalTable	string	No	N/A	Little-endian base64 serialization of a list of x, y floating-point pairs. The x coordinate is the quantized depth value and the y coordinate is the radius of the circle of confusion.	Decode data and update radius dimensions.

Point Cloud

The `PointCloud` element contains properties that provide information regarding the creation and storage of a point cloud.

- The namespace URI is `http://ns.google.com/photos/dd/1.0/pointcloud`.
- The default namespace prefix is `PointCloud`.

Name	Type	Required	Default Value	Property Content	If Container Image Modified
PointCount	integer	Yes	N/A	Number of points (specified by x, y, z, confidence 4-tuples) in the data	No change
Points	string	Yes	N/A	Little-endian base64 serialization of a list of (x, y, z, c) floating-point 4-tuples, where the first three values are the point's XYZ coordinates in the Realm's coordinate system, and the fourth value is the confidence value. Points are in the order: [X1, Y1, Z1, C1, X2, Y2, Z2, C2, ...] Please see the DEPTH_POINT_CLOUD definition in Android's ImageFormat.	No change
Metric	boolean	No		Whether the Position values are expressed in meters. If set to false or not set, the units are unknown (i.e., the point cloud is defined up to a scale). If this value is not set, then some cases (such as measurement) will not be possible.	No change

Appendix

EXIF Consistency

The orientation and aspect ratios of the primary image and depth image will be consistent with the values in the image's EXIF tags, by definition of this specification. Here is a description of this consistency.

The primary image, which is the Container image in most cases, will have the same orientation as the value specified in the EXIF tags. Depth Photos, whose Camera elements do not contain a CameraPose, must specify an Image element. If this Image element is the primary image (and is therefore the Container image), the Depth image will by definition have the same orientation and aspect ratio as the primary image. Otherwise, any other image must be locked to the Device's EarthPose (as the Pose will be zero by definition) and therefore will have the same orientation and aspect ratio as the values in the EXIF tags. In this case, the Depth image will also have the same values.

For 3D use cases such as AR Photos, the EXIF orientation is consistent with the CameraPose in Camera 0, and would apply only to that primary image. For example, suppose that a device is rotated horizontally when taking a photo. Then this orientation would be captured both by the CameraPose, and by the "Horizontal" value in the EXIF orientation field. All other Camera elements would have different CameraPoses, so their respective Image elements would not use the EXIF orientation value, but would instead use the CameraPose specified in their respective parent Camera elements.

Coordinate systems

Earth Coordinate System

Please see the Earth Coordinate System section above.

Realm Coordinate System

Please see the Realm Coordinate System section above.

Camera Coordinate System

The camera coordinate system is the same for all cameras, regardless of the device.
The pose of each camera specifies its position relative to the Realm origin, and its rotation relative to the Realm coordinate system.
Information on camera positioning is available from the device manufacturer, or possibly in a configuration file on the device. Android applications can obtain the camera pose with respect to the Realm coordinate system (realm_T_camera) by doing the following:

AR Applications

- The camera pose is provided by the API. This is Camera.getPose() in ARCore.

Android Applications, non-AR

- Obtain the camera pose (camera_T_device) with respect to the device coordinate system:
  - Android P: Use LENS_POSE_ROTATION and LENS_POSE_TRANSLATION in conjunction with LENS_POSE_REFERENCE
  - Any API prior to P: Use LENS_POSE_ROTATION and LENS_POSE_TRANSLATION
- The concept of a device pose, with respect to the Realm coordinate system (realm_T_device), does not apply for non-AR applications. As such, we can consider that transformation matrix to be the identity matrix.
  Consequently, realm_T_camera is equivalent to device_T_camera.

It may be tempting in a simple single-camera application to use zeroes for the camera positions, implying that the camera is at the device origin. But obviously this would not work in an application that uses multiple cameras, such as depth photography, especially if the image is captured at a close distance (say, a few yards or less).
The following definitions are from the point of view where one is looking out from the camera into the field of view.

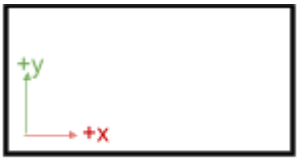
Origin	The center of the image sensor's field of view
Orientation	This is the same as the Android Sensor Coordinate System. The following directions references a setting where the device is held in its native orientation with the screen facing the user. • X is aligned with the horizon, with +X pointing to the user's right hand side. • Y is aligned with gravity, with +Y pointing towards the sky. • Z is perpendicular to the user and is parallel to the camera's optical axis, with +Z pointing towards the user.
Units	Meters
Range	Unbounded
Precision	Single

Image Coordinate System

This coordinate system is used for the image itself, and it uses the Android screen coordinate system. One example usage in this specification is the principal point in PerspectiveModel.

Image Coordinate System

Developers are responsible for transforming objects' vertices into this coordinate system for rendering purposes. Please refer to the OpenGL conventions of model-view-projection (MVP) transformation. The Appendix has further examples available to demonstrate how elements of this specification can be used in parts of this MVP transformation process.



OpenGL Image Coordinates

Device Orientations. The device rendering the image may rotate. As such, the image's coordinate system may need to rotate as well, depending on the application use case. Developers are responsible for doing this transformation from the original to the new rotated coordinate system. To describe this transformation, we first define two subcategories of the Image coordinate system.

Camera Image Coordinate System. This is used for images obtained directly from the camera, with orientation defined by the pixel readout order from the camera sensor. Note that this is often different from screen orientation.

Screen-Oriented Image Coordinate System. This is used for camera images that are rotated to match the rotated device's screen.

Screen Rotations. This section describes how to transform from the Camera Image Coordinate System to the Screen-Oriented Image Coordinate System.

Rotate clockwise by Android Camera2 CameraCharacteristics.SENSOR_ORIENTATION + Device quantized z rotation (around z-axis).

This means that if a phone is held vertically, and the SENSOR_ORIENTATION says "90", then the Screen-Oriented Image Coordinate System is equal to the Camera Image Coordinate System rotated clockwise by 90 degrees. If the phone is then 90 degrees clockwise so it's horizontal (and the UI is allowed to rotate to respect gravity), then the Screen-Oriented Image Coordinate System is equal to the Camera Image Coordinate System rotated clockwise by 180 degrees.

Image Coordinate System Definition. The following assumes a viewer's perspective when facing the image. That is, a person is looking head-on at the image.

Origin	The upper left corner
Orientation	• X is aligned with the long side of the image, with +X going towards the right hand side. • Y is aligned with the short side, with +Y going downwards. • Z is undefined in this 2D coordinate system.
Units	Pixels
Handedness	Right-handed (recommended)
Range	Unbounded
Precision	Single

Object Coordinate System

Please see the Object Coordinate System section above.

Dynamic Depth Poses

All elements' poses are with respect to the Realm coordinate system. The exception is Device:RealmPose, which stores the Realm pose with respect to the Earth, Consequently, transformations between arbitrary coordinate systems can be performed.

Here is an example of how to obtain the pose of camera *i* with respect to the coordinate system of camera *j*, cj_T_ci.

```
ci_T_realm = Inverse(realm_T_ci)

cj_T_ci = cj_T_realm * realm_T_ci
```

Below are the definitions of the poses stored in Dynamic Depth, with their semantics and edge cases.

Device:RealmPose

Semantics	The pose of the Realm with respect to the Earth.
Transformation	earth_T_realm

Required	Optional
Example Data Source	Device GPS data
Notes	Applications that do not have a concept of a Realm can consider realm_T_device to always be the identity pose or identity matrix.

Camera:CameraPose

Semantics	The pose of the Camera with respect to the Realm.
Transformation	realm_T_camera
Required	Yes for the ARPhoto profile.
Example Data Source	AR: Please see Camera.getPose() Android: If LENS_POSE_REFERENCE is the gyroscope, this is just LENS_POSE_TRANSLATION. For Android APIs prior to P, or if LENS_POSE_REFERENCE is the primary camera, add LENS_POSE_TRANSLATION to that of the primary camera. LENS_POSE_ROTATION should always be used directly.
Notes	Applications that do not have a concept of a Realm can consider realm_T_device to always be the identity pose or identity matrix. In this case, realm_T_camera is simply the pose of the camera with respect to the gyroscope on the device. Please see LENS_POSE_REFERENCE.

Plane:PlanePose

Semantics	The pose of the Plane with respect to the Realm.
Transformation	realm_T_plane
Required	Yes, for each Plane in the Planes list.
Example Data Source	ARCore's Plane API. This field does not apply for non-AR applications.
Notes	By definition of planar surface semantics and its usage in AR, the concept of a Realm must be defined in this application.

AppInfo Pose Guidelines

For applications that may choose to store additional information and their associated poses, the following conventions are recommended.

Semantics	The pose of the object with respect to the Realm.
Transformation	realm_T_object
Required	Depends on the image writer's usage of AppInfo:Payload.
Example Data Source	ARCore Anchor APIs