


a summary of standards' extensions 

overview of

GEN<i>CAM 3D

About the Presentation



This presentation was created as basis for technical training sessions focused at implementation of the 3D extensions of GenICam and other standards. It may contain subjective opinions or experience of the author and does not necessarily constitute the official standpoint of the GenICam committee.

The GenICam and the related standards are work in progress, the content of the presentation might become out-of-date. Updated or otherwise extended versions of the presentation might be available from the author.

The presentation is provided on an "as-is" basis. The author makes no warranties regarding the information provided and disclaims liability for damages resulting from its use.

Copyright 2015-2020 by Jan Bečvář ([Groget](#))

This work is licensed under the Creative Commons Attribution-NoDerivatives 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nd/4.0/>.



GenICam is the trademark of EMVA. Camera Link, Camera Link HS, GigE Vision, and USB3 Vision are the trademarks of AIA. CoaXPress and IIDC2 are the trademarks of JIA. All other names are trademarks or trade names of their respective owners.

Latest version of the presentation available from https://www.groget.org/publications/genicam_3d.pdf.



1. **GenlCam 3D** - Background
2. **3D Pixel** - PFNC
3. **3D Coordinates** - SFNC
4. **3D auxiliary data** - GenTL
5. **3D on the wire**
6. **3D Model** Reconstruction
7. **Information** Sources





GenICam 3D - Background

Why & how 



Growing number of 3D implementation but no standard for 3D data exchange

Example of the problems faced

- No existing rules for exchange of 3D data between devices and generic applications
- (Ab)use of color pixel formats, vendor specific mapping to real-world units and coordinate systems
- Vendor specific add-on data transferred per-pixel
- Generic applications cannot process the 3D data effectively (in the acquisition engine)
- Extra burden on side of both device and application vendors to reach interoperability



Working group gathered to discuss standardization

- Major vendors with 3D interest, started Q1 2013
- Initially around GenTL and GigE Vision communities
- Identified set of requirements to satisfy versatile needs of different vendors
- Most of the technical discussions captured in discussion forum genicam.mvtec.com/trac/genicam/discussion/topic/55
- Progress through series of conference calls and committee meetings



The working group identified several orthogonal areas requiring extensions to introduce 3D awareness.

The orthogonality allowed to progress with individual topics independently in subgroups with different pace

- Introduce the notion of a 3D "pixel" → PFNC extensions
- Define how the raw pixel values map to real-world coordinates → SFNC 3D model
- Allow transmitting the 3D together with other data such as luminance, confidence, scatter → GenTL multi-part buffer model
- Transmit the multi-part data effectively considering specifics of individual transfer technologies → GigE Vision protocol extensions (similar for other technologies)



GenDC

- Standardized “Generic Data Container”
- Uniform data description on any TL and/or in a file
- Data structure almost identical with multi-part (except the standard header)
- Therefore not implying changes in other parts of the picture, in particular the SFNC 3D model
- Currently (2019) adoption in TL standards in progress
- Covered in the presentation only where principles between multi-part and GenDC usage differ





3D Pixel - PFNC

The notion of a 3D pixel 

The main goals of the PFNC 3D extension

- Introduce new pixel formats to clearly distinguish 2D, 3D and other types of data
- Avoid abuse of color formats for 3D, color and 3D devices have different needs
- Keep the 3D formats abstract, agnostic of properties such as coordinate systems or units → such properties are attached to the data rather through a well-defined SFNC model
- Support linescan vs. areascan, 3D (point cloud) vs. 2.5D (depth map)
- Build on the existing PFNC infrastructure rather than reinvent the wheel
- Stay transport layer agnostic



Letters A, B, C used to express abstract coordinates

- Independent of the coordinate system (Cartesian, spherical or cylindrical)
- Letter C always expresses the depth (Z, Rho)
- Presence of letters in the pixel name corresponds with presence of (1-3) coordinates in the pixel
- All other pixel properties such as data type, number of bits, padding, etc. reuse existing PFNC infrastructure

Examples:

- Depth only: Coord3D_C8, Coord3D_C16
- Full 3D: Coord3D_ABC16_Planar, Coord3D_ABC8



Some 3D devices need to deliver 3D data in floating point format

- Extended PFNC with support for floating point data type (based on ubiquitous IEC 60559:1989, also known as IEEE 754)
- 32-bit and 64-bit floating point formats (C's float/double)
- Highly recommended to avoid specific floating point values, in particular NaN's to avoid performance penalties
- Floating point pixel formats contain the "f" indicator after the bit-depth number

Examples:

- `Coord3D_ABC32f_Planar`, `Coord3D_C32f`



Specific 3D devices might output only 2 coordinates per pixel

- Linescan based 3D devices would output the X-Z coordinates, while Y is implicit based on camera-object movement between scans (analogical for other coordinate systems)
- The Y-value could be computed from scan number or encoder value (beyond scope of PFNC)

Examples:

- `Coord3D_AC16_Planar`, `Coord3D_AC32f`



Typically used together with other (2D or 3D) image plane

- To express level of validity of corresponding pixels in the "master" image plane
- To cope with laser occlusion, uncertain time-of-flight depth computation etc.
- 1-bit confidence to mark pixels valid or invalid
- Multi-bit int confidence to express level [0, max_value]
- Float confidence to express level, typically [0.0, 1.0]
- Primary use case is 3D, but can be reused eg. for masking non-rectangular images

Examples:

- Confidence1, Confidence1p, Confidence8, Confidence32f





3D Coordinates - SFNC

Mapping the pixel values to real-world coordinates 



The main goals of the SFNC 3D model

- Make the 3D data stream self-describing same as usual in 2D (the protocol itself does not carry sufficient info)
- Stay transport layer agnostic
- Stay independent of the multi-part/GenDC transfers, for devices transferring 3D without additional data multi-part/GenDC is not needed



Fixed, unambiguous model very important for interoperability

- Defined "Scan 3D Control" SFNC chapter
- Observe also "Chunk Data Control", "Encoder Control", and "Image Format Control"
- Guidelines for interpretation of the model given later in this presentation

Devices should aim to implement the model as completely as possible to avoid ambiguity

Chunk data with full description of properties of the 3D data in the buffer should be always attached to the buffer to allow correct interpretation of the data



Enable/disable individual image components and their configuration

- ComponentSelector/Enable (individual components typically transferred as multi-part)
- AOI and PixelFormat control of individual components
- Selecting a "planar" pixel format will result in multiple parts, eg. PixelFormat=Coord3D_ABC8_Planar gives 3 consecutive parts carrying Coord3D_A8, Coord3D_B8 and Coord3D_C8 data (in GenDC that would be a component containing 3 parts)

Note that for complex devices the component handling might be optionally wrapped within RegionSelector and/or SourceSelector



Compromise between effective data transmission (bandwidth) and desired coordinate data range

- It is frequently desirable to transfer the data in integer format for performance reasons
- Conversion to (typically float) real-world coordinates using per-coordinate scale factor and offset



Devices with Cartesian, spherical or cylindrical native coordinates supported

Anchored vs. reference coordinated system

- Reference system marker label on the device

Devices with coordinate system transformation support



Versatile options of 3D data output to support wide range of devices

- 3D vs 2.5D
- Grid vs. unorganized point cloud
- Calibrated, rectified or uncalibrated
- Linescan (Y from encoder or scan number)
- Disparity

SFNC provides clear guidelines how to process data in each of the output modes



Features to query or control which coordinate value (if any) should be treated as invalid pixel mark

Features to assist visualization - definition of bounding box for the data



For additional information refer to

- Chapter "3D Model Reconstruction" later in this presentation
- Detailed descriptions and numerous examples in the SFNC specification





3D auxiliary data - GenTL

Generic model to transfer diverse data together 



The main goals of the GenTL multi-part buffer support

- Allow to deliver various data "carrying the same timestamp" in a single buffer (single event), avoiding problems with identification which data belong together
- Generic description of the target buffer implementable by all underlying transport layers
- Simple model not imposing excessive restrictions for the TL's
- The individual parts might use different data types
- Allow using custom data types as well
- Cover also other related use cases



GenDC (Generic Data Container) aims to solve the same problems as multi-part

- Instead of through per-part information queries it standardizes self-describing container header (TL independent)
- In future expected to be widely used
- Currently (2019) not yet adopted by TL standards and only beta-version implementations available
- Rest of the presentation currently focuses on multi-part, but most of it applies equally to GenDC



Main use cases of the multi-part buffer:

- 3D devices: 3D, luminance, confidence, scatter, even custom data, all together
- Planar formats: unambiguous description of order and location of the planes within the buffer
- Multiple AOI's: all sensor AOI's from a single exposure together
- Multi-source devices: allow transferring data from multi-source (multi-sensor) device together if the sources work in sync (same trigger)
- Non-rectangular images: allow transferring the image data together with pixel mask (confidence)
- Multispectral? More?



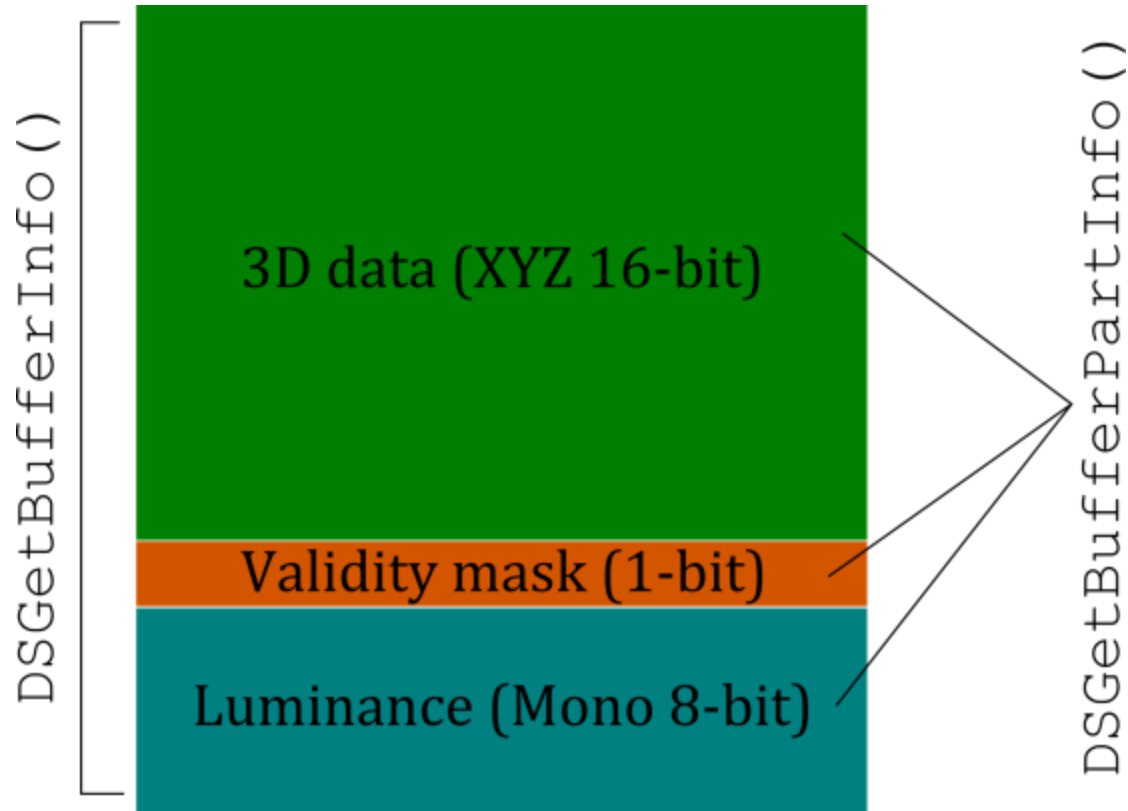
The GenTL buffer can newly optionally consist of multiple distinct parts

- Added new API's to query structure of such buffer and properties of individual parts
- Part info querying analogical to traditional buffer info queries
- If needed, additional information about the payload can be delivered through chunk data
- Order of the parts not significant (except as for planes of the same planar-format entity)
- No need for contiguous structure (padding allowed, helps for variable-size linescan use cases)

GenDC note: in GenDC the data and its properties is not queried through GenTL calls but by parsing the GenDC container header



Multi-part Buffer Principles (cont.)



Is given buffer carrying multipart data?

- `DSGetBufferInfo(BUFFER_INFO_PAYLOADTYPE) == PAYLOAD_TYPE_MULTI_PART`

How many parts are in the buffer?

- `DSGetNumBufferParts()`
- Can be zero (chunks only)
- Beware - unlike in GEV, chunk section is not necessarily reported as a separate part

What is address, size, data type and other properties of the parts?

- `DSGetBufferPartInfo()`



Structure of the parts in the buffer is described by

- Part's base address and data size (actually transferred size)
- Data type (3D image, 2D image, ..., custom)
- Data format (usually PFNC based)
- AOI
- Source ID
- Expected future extensions: region ID, data purpose ID

For multi-part buffer, part-specific properties must be always queried using `DSGetBufferPartInfo()` rather than global `DSGetBufferInfo()`



Chunk data can be appended to buffer with multi-part payload same way as to any other payload type

- For 3D use cases the chunk data is practically inevitable (refer to SFNC section)
- Check chunk data presence using
`DSGetBufferInfo(BUFFER_INFO_CONTAINS_CHUNKDATA)`

GenDC note: GenDC defines its own rules for chunk data handling, the chunk data should be searched and parsed based on the container header, without help of GenTL.



Query which source the data belong to

- Multi-part: BUFFER_PART_INFO_SOURCE_ID
- GenDC: ComponentHeader::SourceId
- Same ID means same source (such as sensor)
- Relate to SFNC using SourceSelector/SourceIDValue and their chunk counterparts

Beware: parts carrying same source ID might not be always pixel-mappable, in particular if the device mixes different types of regions (see SFNC RegionSelector)

Possible reuse by complex devices providing different kinds of data (such as different 3D config, two laser lines, ...) from different parts of the same physical sensor



Analogical to part-source mapping

- Multi-part: BUFFER_PART_INFO_REGION_ID (since v. 1.6)
- GenDC: ComponentHeader::RegionId
- Same ID means same region (and same offset/size parameters)
- Relate to SFNC using RegionSelector/RegionIDValue and their chunk counterparts
- Beware: since SFNC 2.3 region does not mean region of interest, but can have wider meaning (raw sensor output vs. result of processing)



Means to further identify actual purpose of the data and distinguish different parts based on the same basic data_type (such as 2D image)

- Multi-part: BUFFER_PART_INFO_DATA_PURPOSE_ID (since v. 1.6)
- GenDC: ComponentHeader::TypeId
- Values of the ID's are device specific, actual meaning is attached to them from the nodemap - allows easy custom extensions
- Relate to SFNC using ComponentSelector/ComponentIDValue and their chunk counterparts
- In future the SFNC model can be extended/altered for specific use cases?
- Note: since version 2.5 SFNC lists fixed ID's for certain component types



Pixels in a 3D point cloud output might not be organized in a rectangular matrix, just unorganized set of pixels

Multi-part:

- Report width as 1, height as total number of pixels
- Allows to reuse height to report variable size image as known from line scan applications
- Same convention should apply to all transfer technologies

GenDC:

- Use `PartHeader::HeaderType == GDC_1D`
- Considers only one dimension of the data (no x/y)



Currently no mechanism defined to negotiate awareness of the multi-part buffer support between producers and consumers, allowing smart fallback

- The general introduction of "capabilities" into the GenTL standard deferred to a next version (further discussion needed)





3D on the wire

Multi-part implementation in the transfer technologies 



The main goals of the GenTL multi-part buffer proposal

- 100% compatibility with GenTL multi-part buffer as target output
- Stay symmetric with existing GigE Vision conventions wherever possible
- Allow effective linescan-like devices sending the data as available with low latency and low buffering
- Still allow to send the data in simple top-down manner by simple devices
- Allow scalability to higher number of parts in the payload
- Re-use the mechanism to improve other use cases (planar formats, multi-AOI)
- Intended to transfer data carrying common timestamp (while in linescan case it's the common "virtual frame" timestamp)



Designed as extension of the multi-zone payload type from GEV 2.0

- Re-use of principles already existing in the specification
- All the data belonging together over single stream, in single block
- Allows to send data for individual parts in parallel as they are acquired → low latency and low buffering for linescan devices
- But sending the data "top-down" as they will appear in the final buffer is fully valid option
- Allows early drop of packets from parts of no interest by receivers



Leader/trailer carry essential info about the data

- Leader defines structure of the parts in the payload (similar like GenTL multi-part buffer)
- It is up to the receiver if it will store the data to a common GenTL-like buffer or process them separately
- Essential info about all parts (similar as GenTL part information, extended info available through SFNC)
- Trailer carries info about actual data size in each part (allows to cut parts short for variable size linescan use cases)

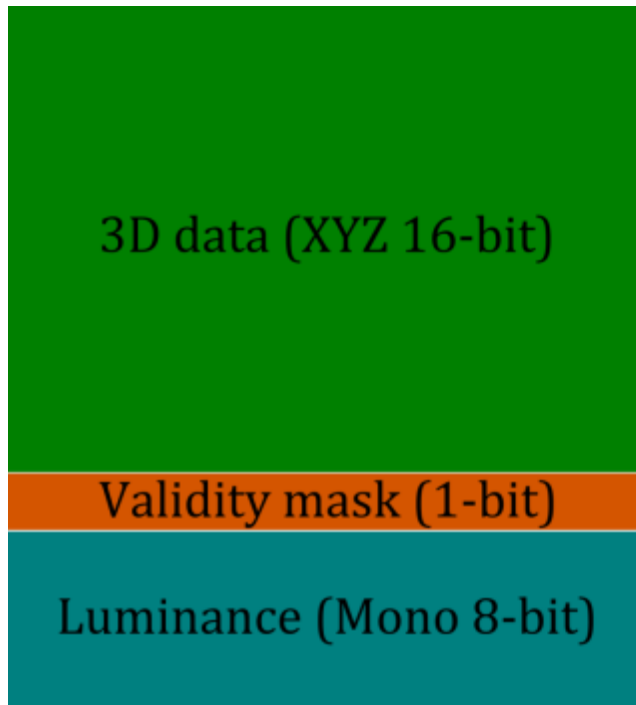


Main rules for streaming the payload packets

- Packets from the same part (and zone) always sent in order
- Packets from different parts can (but do not have to) be sent out of order (arbitrary interleaving)
- Each packet "knows" its offset in the payload
- Packet ID always sequential to allow effective resend
- All packets full-sized except for last packet in a part/zone which can be smaller
- Single packet cannot carry data from multiple parts



Example of possible packet transmission (without zones)



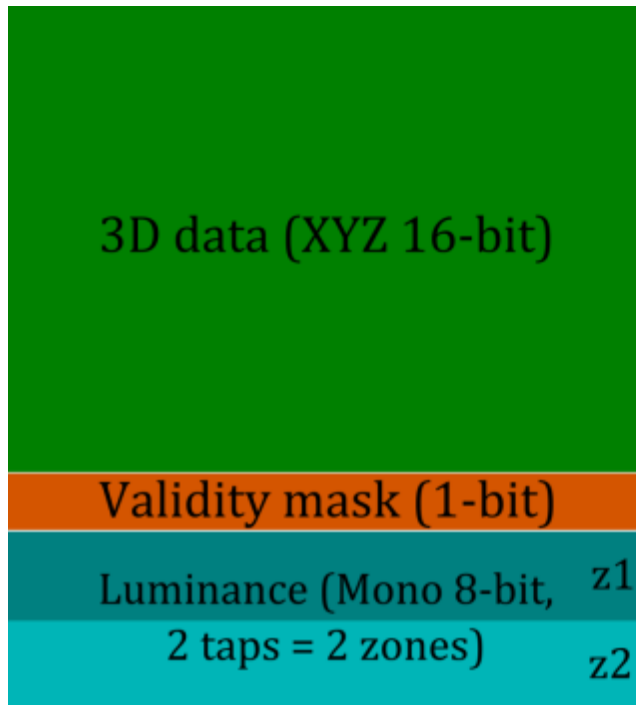
Packets on the wire:

3D_1 V_1 L_1 3D_2 V_2 L_2

or even (consider different sizes):

3D_1 3D_2 L_1 3D_3 V_1 L_2

Example of possible packet transmission (including zones)



Packets on the wire (example):



Various details about GigE Vision multi-part implementation:

- Introduces dedicated payload type and packet format
- Extended ID mode mandatory for multi-part payload type
- Backward compatibility through capability/enable bits
- Each part can be optionally split to multiple zones similar as with existing multi-zone payload (not all receivers might implement that, however)
- All-in packet not possible for multi-part (at least one packet required for each part)



Leader packet size (default 576B) is max limiting factor

- 20B IP header, 8B UDP header, 20B GVSP header, 12 additional "general purpose" leader bytes → 516B left for parts description
- Single part description uses 48B → space for max 10 parts in the standard leader
- Introduced new capability/configuration bit allowing to use larger (up to SCPSx) leader and trailer packets than the default max of 576B
- Controlling entity can only enable this on the transmitter if the receiver also has this capability
- New status code to indicate that the current leader/trailer size is insufficient to carry the entire payload

Chunk data can be appended to the multi-part payload same way as to any other payload:

- Using an extended chunk mode introduced in GEV 2.0
- Only a single set of chunk data common for all parts
- Chunk data travel through the protocol as an additional part (this part does not need to be explicitly reported over GenTL)
- Beware - the extended chunk mode fields in the trailer are appended after all the part description data (offset depends on number of parts)



The streaming related book-keeping can be too high for some transmitter HW designs

- Smaller end-of-part packets, payload gaps in cut-short linescan data → packet resend book-keeping might be non-trivial
- It is therefore valid approach to "reserve" gaps between parts to avoid smaller last packet (will contain padding)
- It is also valid to transfer all the "reserved" packets, even if the actual payload was cut short - such packets would carry only padding
- The transmitter has to report actual data size in the trailer, the receiver has to ignore any padding beyond that
- It is also valid to transfer the data top-down → implies no latency and buffering optimization, no zones

GenTL interface is transport layer agnostic

- GigE Vision specific properties will be hidden and configured within the producer
- Some of the GigE Vision specific options will disappear in the GenTL interface
- In particular, the GenTL Consumer will never see the early per-part data with low latency - in GenTL the buffer is always delivered when complete



Currently (2019) the discussions still in progress

GigE Vision

- Inclusion in version 2.2 close to ready?
- Random access packet addressing (flows not strictly needed)
- Preliminary and final descriptor sharing the same location

USB3 Vision

- Preliminary drafts
- Bulk streams to handle flows?
- Preliminary and final descriptor in leader/trailer





3D Model Reconstruction

Hints on the acquired data processing 



Understanding and correctly implementing the SFNC model for 3D data exchange is key for interoperability

- Many parameters needed that are not provided in the basic stream, additional info needed through SFNC
- SFNC "Scan 3D Control" and related chapters
- Same interpretation by all vendors very important
- Compare and communicate with partners and competitors
- Prefer clarification in public over custom interpretations to keep the model usage consistent



When collecting parameters about the buffer contents, use the possible information sources in following order:

1. Stream protocol (buffer infos for GenTL, protocol headers "on the wire") - most straightforward, always available, unlikely to fail
2. Chunk data - better than nodemap features because it is bound to given buffer
3. Nodemap - only last resort approach, the "current" device configuration might not match the buffer contents any more
4. Have some reasonable defaults wherever applicable for devices not providing all the parameters



Following slides provide hints on possible 3D data processing approach, helping to understand the model (main points, no optimization):

1. Identify Sources of Data
2. Read Pixel Data
3. Consider Pixel Validity
4. Scale Raw Coordinates
5. Linearize the Coordinates
6. Convert Coordinate System
7. Transform Coordinate System
8. Units Conversion
9. Build the 3D Model

The algorithm is interesting also for device vendors to clarify how the parameters will be actually used



Step 1: Identify Sources of Data



Some of the input data might be available from different sources

- Look which data parts are present in the payload and their format
- Compare this info with value of ChunkScan3dOutputMode
- The depth coordinate C should be explicit in the payload
- Coordinate A explicit in the payload or implicit from pixel position?
- Coordinate B explicit in the payload, based on ChunkEncoderValue (linescan) or implicit from pixel position?
- Pixel confidence explicit in the payload or encoded through "invalid values" (ChunkScan3dInvalidDataFlag/Value) within the coordinates or no confidence available (all pixels valid)?
- Is color/luminance data present?
- Are data with other (custom) pixel properties present?

Step 2: Read Pixel Data



Loop through all pixels in the payload, reading the coordinate, confidence and other values for each pixel

- Read the "transferred" data from locations identified in previous step
- The raw coordinate values transferred by the device are abstract (A, B, C), no assumption about the coordinate system at this stage

```
foreach : pixel
  read (transferred_a)
  read (transferred_b)
  read (transferred_c)
```

(etc.: confidence, color, ...)



Step 3: Consider Pixel Validity



Skip pixels carrying invalid pixel flag (ChunkScan3dInvalidDataFlag/Value)

If desired, consider pixel validity based on the known confidence value

- Drop pixels under desired threshold if desired
- The actual treatment of pixel confidence is application specific

```
if (transferred_confidence < THRESHOLD)
    continue
```

(just example)



Step 4: Scale Raw Coordinates



Scale the raw transferred coordinate values

- The scale/offset is reported for each coordinate in `ChunkScan3dCoordinateScale`, `ChunkScan3dCoordinateOffset`
- If missing, assume `scale=1`, `offset=0`

```
scaled_a = transferred_a * SCALE_A + OFFSET_A
```

```
scaled_b = transferred_b * SCALE_B + OFFSET_B
```

```
scaled_c = transferred_c * SCALE_C + OFFSET_C
```



Step 5: Linearize the Coordinates



For non-disparity output modes no action needed (already linear)

For disparity, the C coordinate is inversely proportional to the actual range

- Read Scan3dBaseline and Scan3dFocalLength
- Read Scan3dPrincipalPointU/V (default image center)
- Read all with ChunkComponentSelector selecting Disparity

```
linear_a = (scaled_a - PRINPT_U) * BASELINE / scaled_c  
linear_b = (scaled_b - PRINPT_V) * BASELINE / scaled_c  
linear_c = FOCAL_LEN * BASELINE / scaled_c
```

Step 6: Convert Coordinate System



Convert the data to the target coordinate system (typically Cartesian)

- Source device coordinate system identified by `ChunkScan3dCoordinateSystem`
- If missing, assume right-hand Cartesian
- From this point the abstract coordinates A, B, C become Cartesian X, Y, Z

`cartesian_x = (depends on source coord. system)`

`cartesian_y = (depends on source coord. system)`

`cartesian_z = (depends on source coord. system)`

Step 7: Transform Coordinate System



Optionally perform automatic coordinate system transformation

- If the data are in the "anchor" coordinate system of the device (ChunkScan3dCoordinateSystemReference==Anchor)
- Parameters to transform from native anchor system to the reference one: ChunkScan3dCoordinateReferenceSelector/Value
- Observe the reference point marker at the device
- Useful eg. for intuitive live-view from the "sensor perspective", regardless the device type
- Applied in following order: rotation along x, rotation along y, rotation along z, translation

```
xrotated_x = cartesian_x  
yrotated_x = xrotated_x*cos(ROT_Y) + xrotated_z*sin(ROT_Y)  
zrotated_x = yrotated_x*cos(ROT_Z) - yrotated_y*sin(ROT_Z)  
transformed_x = zrotated_x + TRANS_X  
(similar for y and z coordinates)
```



Step 8: Units Conversion



If needed, convert the data to the correct distance units

- Distance units used by the device queried via `ChunkScan3dDistanceUnit`
- If missing, assume millimeters
- For disparity always meters implied by SFNC model

```
output_x = transformed_x / UNIT_SCALE_FACTOR
```

```
output_y = transformed_y / UNIT_SCALE_FACTOR
```

```
output_z = transformed_z / UNIT_SCALE_FACTOR
```

Step 9: Build the 3D Model



In previous steps the 3D coordinates of all pixels were computed

- Use them to build the 3D model in desired application specific format
- Optionally attach color/luminance or other info (observe per-component binning/decimation, disparity-based devices often output range and luminance data in different resolution)
- `ChunkScan3dAxisMin/Max` can give hints about the bounding box of the 3D data



The shown algorithm is just an example that might not be fully applicable in all cases (some steps optional)

Advanced topics such as multi-source devices not considered in the algorithm

A real-world implementation would require various optimizations and fine algorithm control through user parameters





Information Sources

How to stay in sync 



SVN-Trac-Forum-Wiki genicam.mvtec.com/trac/genicam

- Trac forum with most of the 3D working group discussions
genicam.mvtec.com/trac/genicam/discussion/topic/55
- Trac ticket about 3D SFNC extensions (and other related tickets)
genicam.mvtec.com/trac/genicam/ticket/1221
- Lots of info that can help understand the final standards

Official website www.genicam.org

- Useful in particular to download all official releases and minutes from technical meetings

Mailing list genicam@list.stemmer-imaging.com

- Official place to ask for help
- Many of technical discussions (even historical - archive)

Groget 😊 (www.groget.org, incl. [GenICam introduction](#) presentation)



Framework for implementation of GenTL Producers

- Contains “Viky” – a toy GenTL Producer to demonstrate the framework usage and to allow direct experiments
- Includes simple demonstration of 3D output with everything required to properly interpret it (e.g. the chunk data implementing the 3D model parameters)
- Could provide useful hints even for developers not directly targeting GenTL
- Available (full source code) to GenICam group members from the GenICam SVN repository



Summary of versions of individual standard documents where 3D awareness was or will be introduced

- PFNC 2.0 (part of GenICam 3.0)
- SFNC 2.2 (part of GenICam 3.0)
- GenTL 1.6 (GenDC & multi-part, part of GenICam 2019.11)
- GenTL SFNC 1.1
- GigE Vision 2.1 (multi-part), 2.2 (GenDC, not yet ratified)
- GenDC 1.0 (part of GenICam 2019.11)
- CoaXPress?
- CameraLink HS?
- USB3 Vision?



GenICam group membership

- Is free (for companies/organizations)
- Provides access to the group's resources and knowledge base

How to join:

- On www.genicam.org navigate to page named "GenICam Group Members" (the actual link might change)
- The page contains section "How to become an associated member?" with corresponding instructions





What else?

Further discussion...

Time to start with actual implementation? 



The main title 'GEN<i>CAM 3D' is centered in the middle section. 'GEN' and 'CAM' are in a bold, black, sans-serif font. The letter 'i' is in a pink, italicized font and is enclosed in pink angle brackets '<i>' and '</i>'. '3D' is in a white, sans-serif font. The background is a dark grey textured rectangle.