Cloud Optimized Point Cloud Specification – 1.0

Figure 1: COPC Logo

Table of contents

Version

This document defines Cloud Optimized Point Cloud (COPC) version **1.0**.

This document is available as a PDF at copc-specification-1.0.pdf.

Introduction

A COPC file is a LAZ 1.4 file that stores point data organized in a clustered octree. It contains a VLR that describe the octree organization of data that are stored in LAZ 1.4 chunks.

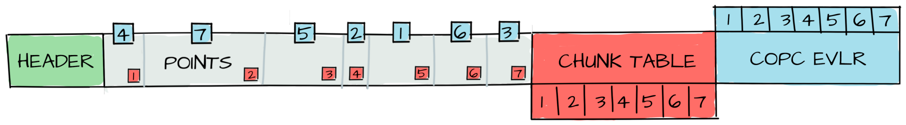

Figure 2: `info` VLR and the LAZ chunk table allow COPC readers to select and seek through the file.

Data organization of COPC is modeled after the EPT data format, but COPC clusters the storage of the octree as variably-chunked LAZ data in a single file. This allows the data to be consumed sequentially by any reader than can handle variably-chunked LAZ 1.4 (LASzip, for example), or as a spatial subset for readers that interpret the COPC hierarchy. More information about the differences between EPT data and COPC can be found below.

Notation

Some of the file format is described using C-language fixed width integer types. Groups of entities are denoted with a C-language struct, though all data is packed on byte boundaries and encoded as little-endian values, which may not be the case for a C program that uses the same notation.

Implementation

Key aspects distinguish an organized COPC LAZ file from an LAZ 1.4 that is unorganized:

- It *MUST* contain *ONLY* LAS PDRFs 6, 7, or 8 formatted data
- It *MUST* contain a COPC `info` VLR
- It *MUST* contain a COPC `hierarchy` VLR

LAS PDRFs 6, 7, or 8

COPC files *MUST* contain data with *ONLY* ASPRS LAS Point Data Record Format 6, 7, or 8. See the ASPRS LAS specification for details.

`info` VLR

User ID	Record ID
`copc`	`1`

The `info` VLR *MUST* exist.

The `info` VLR *MUST* be the **first** VLR in the file (must begin at offset 375 from the beginning of the file).

The `info` VLR is `160` bytes described by the following structure. `reserved` elements *MUST* be set to `0`.

```
struct CopcInfo
{

  // Actual (unscaled) X coordinate of center of octree
  double center_x;

  // Actual (unscaled) Y coordinate of center of octree
  double center_y;

  // Actual (unscaled) Z coordinate of center of octree
  double center_z;

  // Perpendicular distance from the center to any side of the root node.
  double halfsize;

  // Space between points at the root node.
  // This value is halved at each octree level
  double spacing;

  // File offset to the first hierarchy page
  uint64_t root_hier_offset;

  // Size of the first hierarchy page in bytes
  uint64_t root_hier_size;

  // Minimum of GPSTime
  double gpstime_minimum;

  // Maximum of GPSTime
  double gpstime_maximum;

  // Must be 0
  uint64_t reserved[11];
};
```

`hierarchy` VLR

User ID	Record ID
`copc`	`1000`

The `hierarchy` VLR *MUST* exist.

Like EPT, COPC stores hierarchy information to allow a reader to locate points that are in a particular octree node. Also like EPT, the hierarchy *MAY* be arranged in a tree of pages, but *SHALL* always consist of at least ONE hierarchy page.

The VLR data consists of one or more hierarchy pages. Each hierarchy data page is written as follows:

The VoxelKey corresponds to the naming of EPT data files.

```
struct VoxelKey
{
  // A value < 0 indicates an invalid VoxelKey
  int32_t level;
  int32_t x;
  int32_t y;
  int32_t z;
}
```

An entry corresponds to a single key/value pair in an EPT hierarchy, but contains additional information to allow direct access and decoding of the corresponding point data.

```
struct Entry
{
  // EPT key of the data to which this entry corresponds
  VoxelKey key;

  // Absolute offset to the data chunk if the pointCount > 0.
  // Absolute offset to a child hierarchy page if the pointCount is -1.
  // 0 if the pointCount is 0.
  uint64_t offset;

  // Size of the data chunk in bytes (compressed size) if the pointCount > 0.
  // Size of the hierarchy page if the pointCount is -1.
  // 0 if the pointCount is 0.
  int32_t byteSize;

  // If > 0, represents the number of points in the data chunk.
  // If -1, indicates the information for this octree node is found in another hierarchy pag
  // If 0, no point data exists for this key, though may exist for child entries.
  int32_t pointCount;
}
```

The entries of a hierarchy page are consecutive. The number of entries in a page can be determined by taking the size of the page (contained in the parent page as `Entry::byteSize` or in the COPC `info` VLR as `CopcData::root_hier_size`) and dividing by the size of an `Entry` (32 bytes).

```
struct Page
{
    Entry entries[page_size / 32];
}
```

Differences from EPT

- COPC has no ept.json. The information from ept.json is stored in the LAS file header and LAS VLRs.
- COPC currently provides no support for ept-sources.json. File metadata support may be added in the future.
- COPC only supports the LAZ point format and does not support binary point arrangements.
- COPC chunks store only point data as LAZ. EPT, when stored as LAZ, uses complete LAZ files including the LAS header and perhaps VLRs.

Example Data

- The venerable Autzen Stadium file commonly used in PDAL and other open source testing scenarios is available as a 80mb COPC file at https://github.com/PDAL/data/blob/master/autzen/autzen-classified.copc.laz

View it in your browser at https://viewer.copc.io/?copc=https://s3.amazonaws.com/hobu-lidar/autzen-classified.copc.laz

- SoFi Stadium is available as a 2.3gb COPC file at https://hobu-lidar.s3.amazonaws.com/sofi.copc.laz.

View it in your browser at https://viewer.copc.io/?copc=https://s3.amazonaws.com/hobu-lidar/sofi.copc.laz

The data are courtesy of US Army Corps of Engineers Remote Sensing & GIS Center of Expertise / National Center for Airborne Laser Mapping

- Millsite is available as a 1.9gb COPC file at https://s3.amazonaws.com/data.entwine.io/millsite.copc.laz.

View it in your browser at https://viewer.copc.io/?copc=https://s3.amazonaws.com/data.entwine.io/millsite.copc.laz

The data are from the USGS 3DEP Millsite Reservoir Collection

Reader Implementation Notes

COPC is designed so that a reader needs to know little about the structure of a LAZ file. By reading the first 549 bytes (375 for the header + 54 for the COPC VLR header + 160 for the COPC VLR), the software can verify that the file is a COPC file and determine the point data record format and point data record length, both of which are necessary to create a LAZ decompressor.

Readers should: * verify that the first four bytes of the file contain the ASCII characters "LASF". * verify that the 4 bytes starting at offset 377 contain the characters `copc`. * verify that the bytes at offsets 393 and 394 contain the values 1 and 0, respectively (this is the COPC version number, 1). * determine the point data record format by reading the byte at offset 104, masking off the two high bits, which are used by LAZ to indicate compression, and can be ignored. * determine the point data record length by reading two bytes at offset 105.

The octree hierarchy is arranged in pages. The COPC VLR provides information describing the location and size of root hierarchy page. The root hierarchy page can be used to traverse to child pages. Each entry in a hierarchy page either refers to a child hierarchy page, octree node data chunk, or an empty octree node. The size and file offset of each data chunk is provided in the hierarchy entries, allowing the chunks to be directly read for decoding.

Credits

COPC was designed in July–November 2021 by Andrew Bell, Howard Butler, and Connor Manning of Hobu, Inc.. Entwine and Entwine Point Tile were also designed and developed by Connor Manning of Hobu, Inc

Support

COPC development was supported by

CRREL

Microsoft

Pronunciation

There is no official pronunciation of COPC. Here are some possibilities:

- cah-pick – `ka pIk`
- co-pick – `kö pIk`
- cop-see – `kap si`
- cop-pick – `kap pIk`
- see oh pee see – `si o pi si`

Discussion

Use Case

Cloud Optimized GeoTIFF has shown the utility and convenience of taking a dominant container format for geospatial raster data and optionally augmenting its organization to allow incremental "range-read" support over HTTP with it. With the mantra of "It's just a TIFF" allowing ubiquitous usage of the data content combined with the flexibility of supporting partial reads over the internet, COG has found a sweet spot. Its reward is the ongoing rapid conversion of significant raster data holdings to COG-organized content to enable convenient cloud consumption of the data throughout the GIS industry.

What is the COG for point clouds? It would need to be similar in fit and scope to COG:

- Support incremental partial reads over HTTP
- Provide good compression
- Allow dimension-selective reads
- Provide all metadata and supporting information
- Support an EPT-style octree organization for data streaming

"Just a LAZ"

LAZ (LASZip) is the ubiquitous geospatial point cloud format. It is an augmentation of ASPRS LAS that utilizes an arithmetic encoder to efficiently compress the point content. It has seen a number of revisions, but the latest supports dimension-selective access and provides all of the metadata support that normal LAS provides. Importantly, multiple software implementations (laz-rs, laz-perf, and LASzip) provide LAZ compression and decompression, and laz-perf and laz-rs include compilation to JavaScript which is used by all JavaScript clients when consuming LAZ content.

Put EPT in LAZ

The EPT content organization supports LAZ in its current "exploded" organization. Exploded in this context means that each chunk of data at each octree level is stored as an individual LAZ file (or simple blob, or a zstd-compressed blob). One consequence of the exploded organization is large EPT trees of data can mean collections of *millions* of files. In non-cloud situations, EPT's cost when moving data or deleting it can be significant. Like the tilesets of late 2000s raster map tiles, lots of little files are a problem.

LAZ provides a feature that allows concatenation of the individual LAZ files into a single LAZ file. This is the concept of a dynamically-sized chunk table. It is a feature that Martin Isenburg envisioned for quad-tree organized data, but it could work the same for an octree.

Structural Changes to Draft Specification

- Removed `count` from `Page` struct
- Changed Record ID of COPC hierarchy EVLR from 1234 to 1000
- Require reserved entries of the COPC VLR to have the value 0
- Require the COPC VLR to be located immediately after the header at offset 375.
- Increase the size of the COPC VLR data structure to 160 bytes.
- Add `laz_vlr_offset`, `laz_vlr_size`, `wkt_vlr_offset`, `wkt_vlr_size`, `eb_vlr_offset`, `eb_vlr_size` to the COPC VLR, replacing 6 `reserved` entries.
- PDRF must be 6, 7, or 8
- Add `extents` VLR with UserID of `copc` and record ID of 10000.
- VLR UserIDs switched from `entwine` to `copc`
- Removed `laz_vlr_offset`, `laz_vlr_size`, `wkt_vlr_offset`, `wkt_vlr_size,`eb_vlr_offset`,`eb_vlr_size
  `the COPC info VLR. Added 8`reserved‘ entries.
- Describe hierarchy entries for empty octree nodes.
- Add back `root_hier_offset` and `root_hier_size` in COPC info VLR. Removed 2 `reserved` entries.
- Remove `extents` VLR and put gpstime_minimum and gpstime_maximum in `info` VLR.