3D Tiles 1.1

3D Tiles is an OGC Community Standard that as become a vital part of the 3D geospatial ecosystem. 3D Tiles 1.1 is a revision of the original 3D Tiles standard that is fully backward compatible, and adds new functionalities that addresses the needs of the the next generation of geospatial applications.

0. What's New in 3D Tiles 1.1

- **Simpler definition of tile content**: glTF assets can now be stored directly as the tile content. This improves the interoperability with 3D content creation tools, and allows a unified view on a large variety of 3D content, like heterogeneous 3D models, point clouds, or instanced models.
- **More flexibility for organizing tile content**: Each tile can include multiple contents. This allows the same content to be stored and delivered in different formats, and makes it possible to define groups of content across multiple tiles.
- **Implicit tiling**: Tilesets consisting of quadtrees or octrees can be represented in a compact binary form. The implicit tiling scheme enables random access to tiles in the implicit hierarchy, and allows new, more efficient traversal algorithms.
- **Metadata for tilesets, tiles and groups of content:** 3D Tiles 1.1 includes a generic specification for 3D metadata. This metadata can be associated with the core elements of a tileset, on all levels of granularity. The versatile structure and compact storage formats that are enabled with the 3D Metadata Specification make it possible to capture semantically rich data and make it accessible for the end-users of geospatial applications.

1. A Quick Introduction to 3D Tiles 1.0

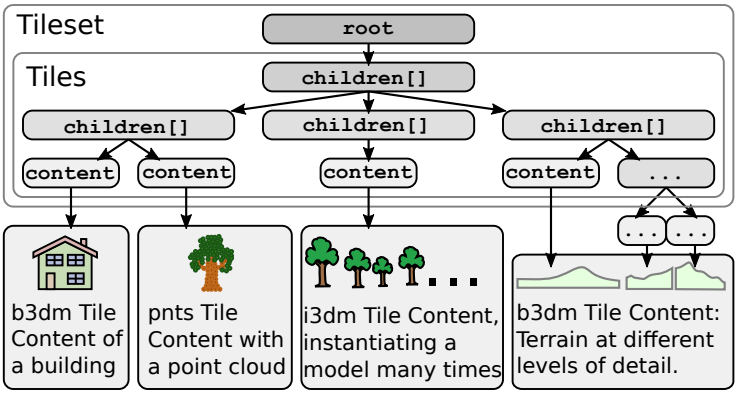
This is a quick summary of core concepts of the 3D Tiles 1.0 standard. If you are already familiar with 3D Tiles 1.0, you can skip this section. An overview of 3D Tiles 1.0 and the full specification are available at https://github.com/CesiumGS/3d-tiles

3D Tiles 1.0 is an open specification for storing, streaming and visualizing heterogeneous 3D geospatial content. A **tileset** is stored as a JSON file that contains a hierarchical structure of **tiles**. Each tile can refer to renderable tile **content**, which is stored in binary files.

Each tile and tile content can have a **bounding volume**. This describes the spatial extent in geographic coordinates, using a bounding region, or in cartesian coordinates, using a bounding box. Together, the bounding volumes form a spatially coherent hierarchy.

Different types of content are supported via different tile formats, such as:

- **Batched 3D Model (b3dm)**: Heterogeneous models like textured terrain or 3D buildings
- **Instanced 3D Model (i3dm)**: Multiple instances of the same 3D model
- **Point Clouds (pnts)**: A massive number of points, for example, photogrammetry data



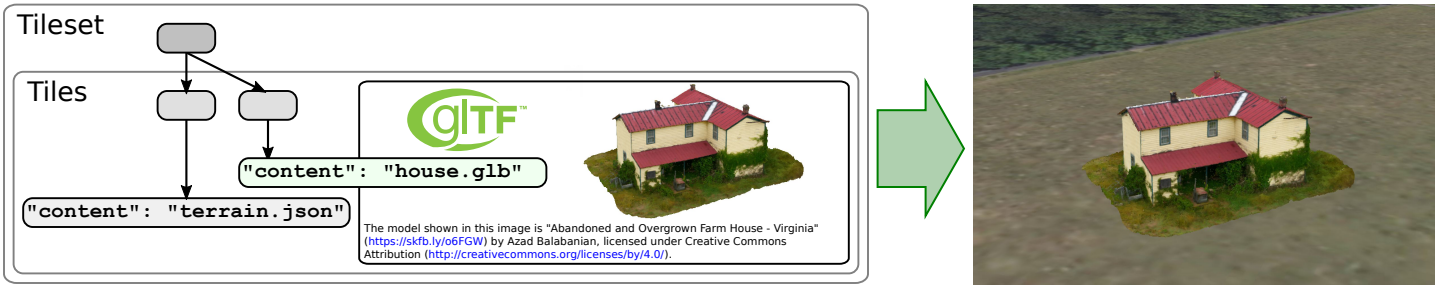
Each tile content can consist of multiple **features**. A single feature may be one part of a model in B3DM content, a point or a group of points in PNTS content, or an instance in I3DM content.

The tile content can also contain a **Feature Table** and a **Batch Table**. These tables store additional data for each feature of the tile content: The Feature Table stores information for rendering the features. The Batch Table stores metadata for each feature.

2. The New Features of 3D Tiles 1.1

2.1. Direct Support for glTF as Tile Content

3D Tiles 1.1 improves the interoperability between 3D Tiles and the glTF ecosystem: glTF assets can now directly be used as the content of any tile.



glTF is becoming the main tile content format for 3D Tiles 1.1. The ubiquitous support for glTF and its robust ecosystem make it possible to deliver tile content to a large variety of clients in a unified way.

Migration of Tile Formats to glTF

The versatility of glTF allows emulating many of the functionalties of the original 3D Tiles 1.0 tile formats directly:

3D Tiles 1.0 tile format feature	glTF equivalent
Relative-To-Center rendering (with `RTC_CENTER`)	The `RTC_CENTER` can be added to the translation component of the root node of the glTF asset
Feature table properties like `POSITION`, `NORMAL`, or `COLOR` for point clouds	Properties for individual vertices (points) can be stored as standard glTF vertex attributes
Constant colors for all points of a point cloud (with `CONSTANT_RGBA`)	Constant point colors be assigned with materials or with equal per-point colors in vertex attributes

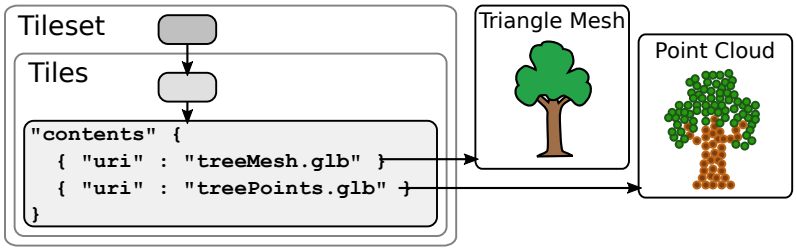
The functionalities that are not directly supported by glTF can be emulated with the appropriate glTF extensions. The following list shows how certain features of the original tile formats can be represented in glTF, and which glTF extensions they require:

3D Tiles 1.0 tile format feature	glTF equivalent
Batch IDs and Batch Tables	The `EXT_mesh_features` extension can be used to assign IDs to mesh features. The `EXT_structural_metadata` extension can be used to associate mesh features with metadata
GPU instancing in Instanced 3D Models	The `EXT_mesh_gpu_instancing` extension allows GPU instancing directly in glTF
Compression for Point Clouds	The `KHR_mesh_quantization` extension offers simple quantization based compression. The `EXT_meshopt_compression` extension allows quantization- and entropy-based compression.

Additional glTF extensions can be used in order to deliver further optimized tile content. For example, the `KHR_draco_mesh_compression` extension can be used to provide glTF assets in an even more compact form, by employing the Draco geometry compression methods. Texture compression is supported via the `KHR_texture_basisu` extension, which reduces the transmission size and GPU memory footprint of textures using KTXv2 images with Basis Universal supercompression.

2.2. Support for Multiple Contents

With 3D Tiles 1.1, it is possible to assign multiple contents to a single tile. This allows more flexible tileset structures: For example, a single tile can now contain multiple representations of the same geometry data, once as a triangle mesh and once as a point cloud:



When combining this functionality with the metadata support of 3D Tiles 1.1, the contents can also be arranged in groups, and these groups can be associated with metadata. This allows applications to perform styling or optimizatons based on the group that the content belongs to.

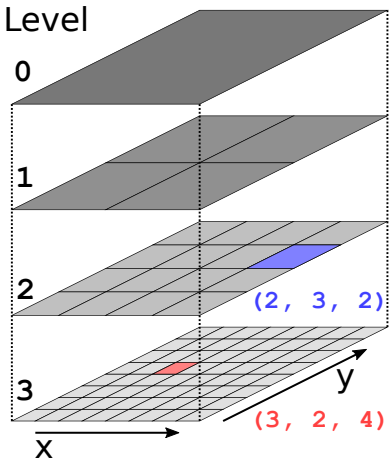
2.3. Support for Implicit Tiling Schemes

When a tileset is represented as a uniform quadtree or octree, then it is not necessary to store the actual tile structure as nested JSON elements. The regular pattern allows a more compact representation of the structure. 3D Tiles 1.1 introduces support for implicit tilesets, including a compact binary format for representing the tile hierarchy. This format offers random access to the available tiles and their content. This enables new traversal algorithms, by allowing a direct lookup of the availability information, without an explicit traversal of the tile hierarchy.

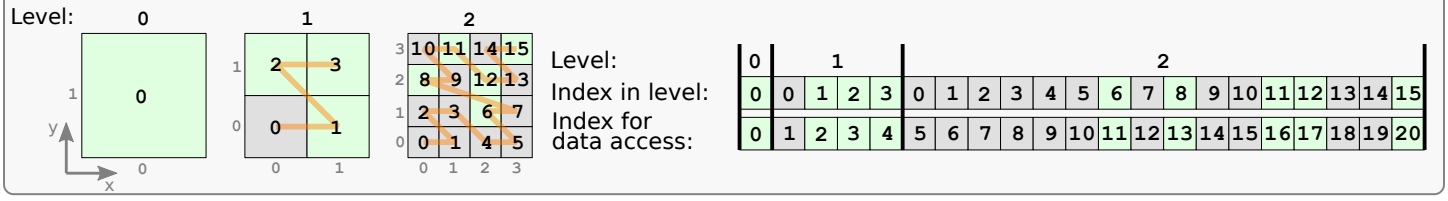
3D Tiles 1.1 supports different **subdivision schemes** for the implicit tile hierarchy: The bounding volume of the root tile is subdivided recursively into four or eight equal-sized parts, forming a **quadtree** or **octree**.

Within this implicit hierarchy, individual tiles can be accessed directly with their local **tile coordinates**: The coordinates of a tile within a quadtree are given as `(level, x, y)`. For an octree, the coordinates are given as `(level, x, y, z)`.

For each tile of the hierarchy, the implicit tiling scheme stores two pieces of information in a binary buffer: The **tile availability** indicates which of the tiles are present, and the **content availability** indicates which of the tiles have content. The tile coordinates can be used to directly access this information.



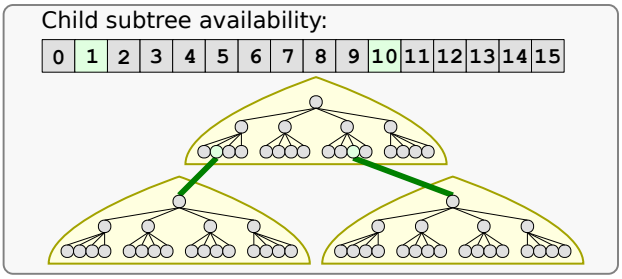
An example of tile- or content availability storage for a quadtree, with available elements indicated by green cells. Due to the regular structure of the hierarchy, the tile coordinates can directly be converted into an index that can be used for accessing the binary data:



The hierarchy that is modeled with implicit tiles can be large. To enable efficient traversal of large hierarchies, the implicit tiles are partitioned into **subtrees**: Each subtree stores information about the structure and availability of content for a fixed-size section of the tileset.

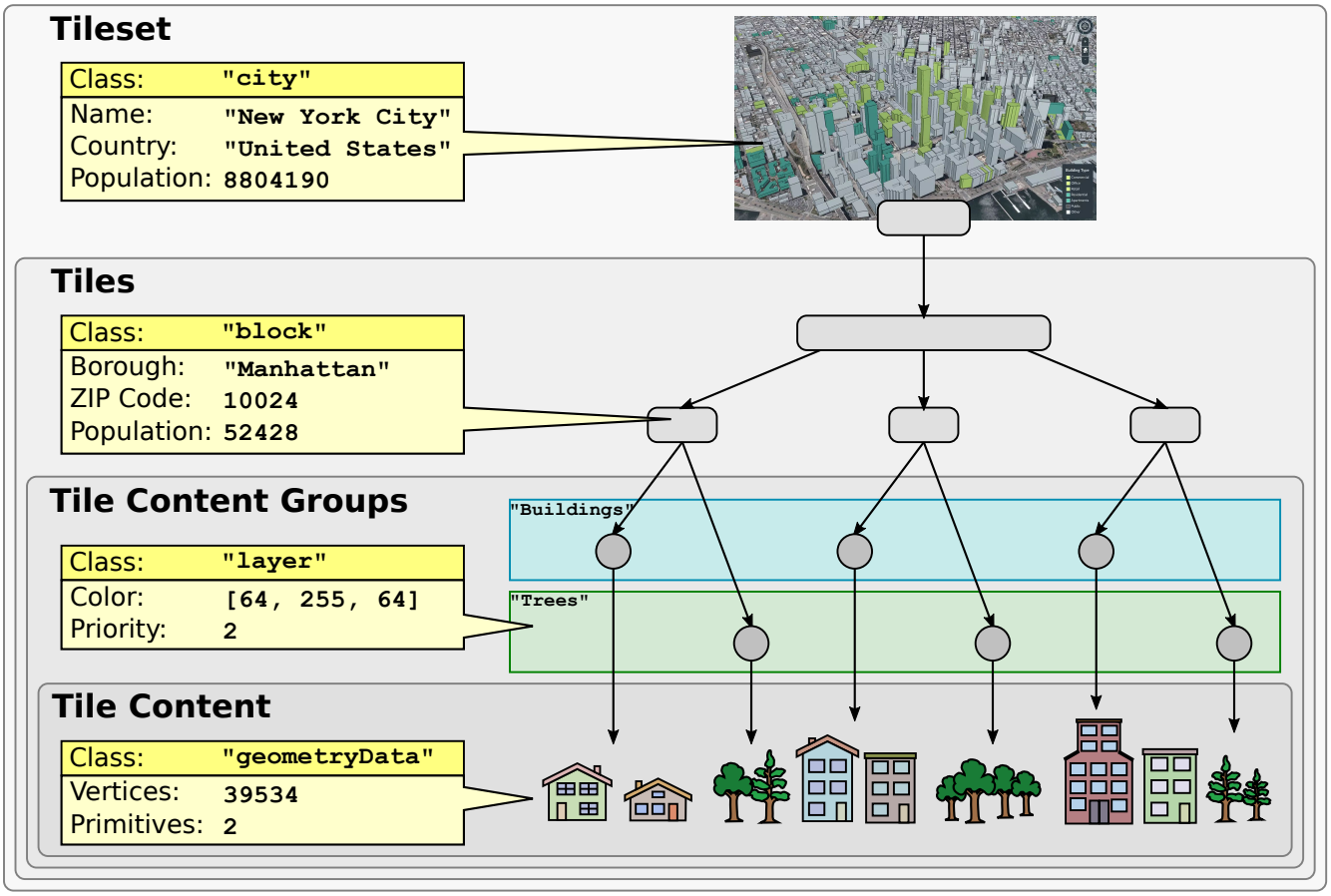
Additionally, the subtree stores information about further subtrees that may be available. This **child subtree availability** is also stored in a binary buffer.

The entire tileset is described by this tree of subtrees. Clients may use the child subtree availability to only request subtrees that are relevant at runtime.

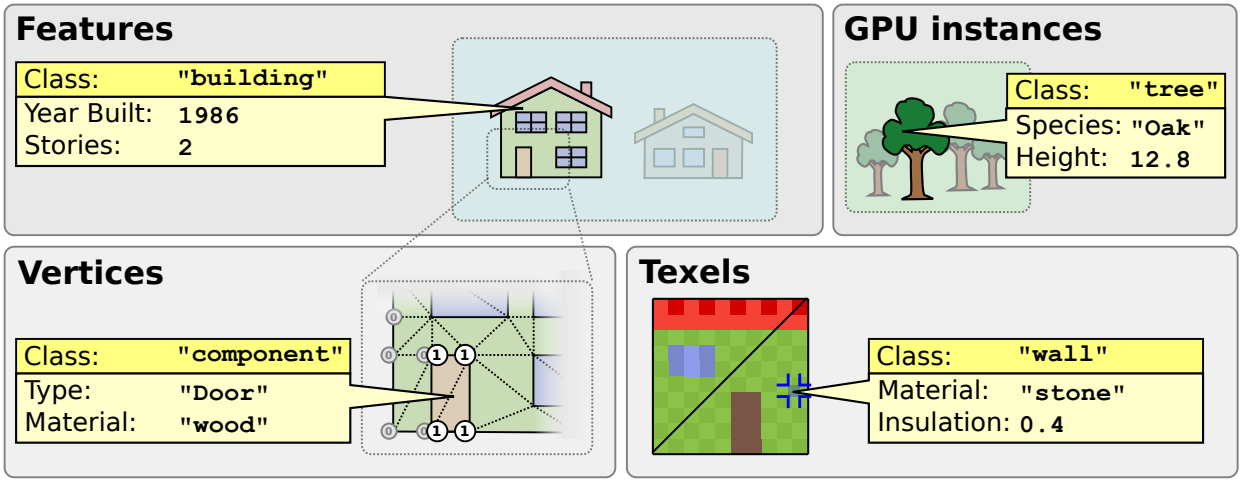


3. 3D Metadata Introduction

The 3D Metadata Specification defines a standard metadata format for 3D data. The specification is language- and format agnostic: It defines key concepts for structured metadata that can be associated with 3D data. 3D Tiles 1.1 defines an implementation of these concepts in the context of 3D Tiles. This allows associating metadata with 3D data on all levels of granularity within a tileset:



The specification can also be leveraged in other 3D formats. For example, the **`EXT_structural_metadata`** extension is a proposed extension for glTF which allows associating 3D metadata with glTF tile content on an even finer level of granularity:



4. The 3D Metadata Specification

The 3D Metadata Specification defines a standard format for structured metadata. The core concept is the definition of a schema for the data. This schema is language- and format agnostic: It does not impose a particular format for the serialization of the schema itself, or for the storage of the actual data. Instead, it can be combined with the storage format that is most suitable for the level of granularity on which the metadata is applied.

4.1. Metadata Structure Definition

The structure of metadata is defined with a **schema**. Each schema consists of a set of **classes** and **enums**. A class consists of a set of **properties**, where each property has a certain type. The type can be one of multiple primitive- or structured types, or an enum type, meaning that the actual value of the property is one of multiple, enumerated, named values.

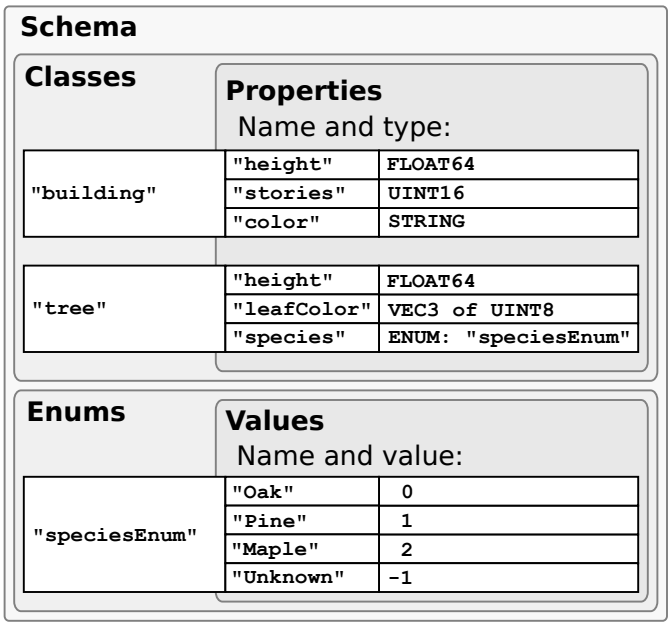
The primitive types include numeric types - namely, integer- and floating-point types with different sizes - as well boolean, enum types and strings. Fixed- and variable length array types are defined based on these primitive types. The type system also includes dedicated types for 2D, 3D, and 4D vectors and matrices with numeric components.

The properties that are defined in the schema do not have an inherent meaning. But properties may have a **semantic** identifier that allows assigning an application-specific meaning to these properties. This identifier can then be used to look up the semantics of a property in an external semantic reference.

4.2. Storage of Metadata Entities

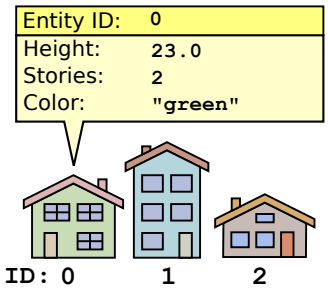
The schema itself only describes the structure of the metadata and the types of properties. But it does not define how the property values are stored. This allows reusing the schema definition across different assets and file formats, and combining it with different storage formats.

A class from the schema can be instantiated, to form a metadata **entity**. Such an entity can be created from a set of property values that conform to the structure of the class. The following diagram shows the case where the metadata values are stored in tables, one for each class, and the entities are created by looking up the property values in these tables:



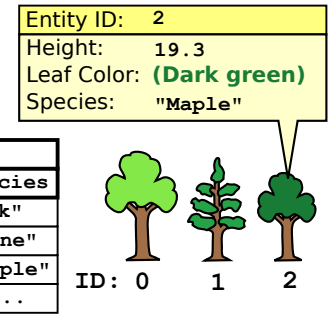
The columns and their types are determined by the schema:

Buildings			
ID	height	stories	color
0	23.0	2	"green"
1	31.8	3	"blue"
2	16.2	2	"brown"
3	...	...	...



The ID of an object is used to look up the metadata values in the corresponding row:

Trees			
ID	height	leafColor	species
0	29.1	[133,233,72]	"Oak"
1	25.8	[37,163,72]	"Pine"
2	19.3	[25,124,54]	"Maple"
3	...	...	...



4.3. Predefined Metadata Storage Formats

Two different ways of encoding metadata values in tabular form are defined in the specification:

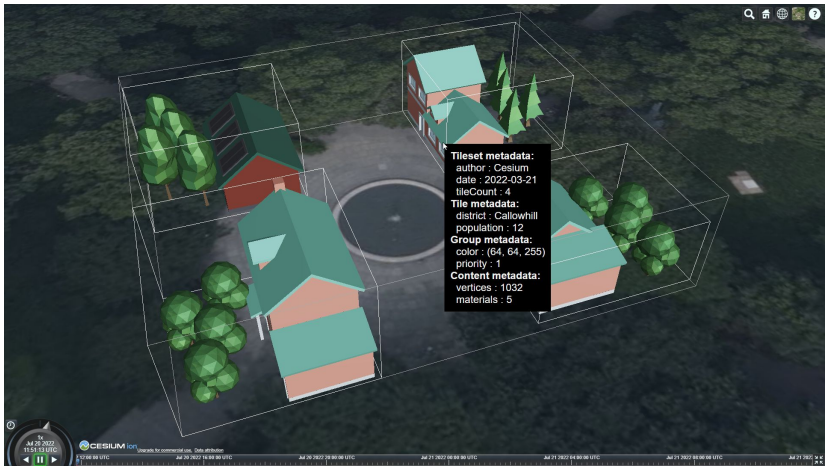
- **Binary Table Format**: Property values are stored in parallel 1D arrays, encoded as binary data
- **JSON Format**: Property values are stored directly in JSON objects

The specification contains further details about the exact encoding and serialization formats. Implementations of the 3D Metadata specification may define their own, custom serialization- and storage formats.

5. Metadata Support in 3D Tiles 1.1

3D Tiles 1.1 includes an implementation of of the 3D Metadata specification that allows assigning metadata to elements of a tileset. By assigning metadata to tilesets, tiles, groups of tiles, or the tile content, different use-cases are supported:

- Metadata information can be displayed for the end-user in a UI.
- Applications can use the metadata to show or hide certain tiles or groups of tiles
- The rendering can be styled based on color- or layer information from the metadata
- Metadata can be used to optimize request patterns, for example, by only requesting certain types of content, or content that has specific tags in its metadata.
- The per-tile metadata can be used to optimize traversal algorithms.



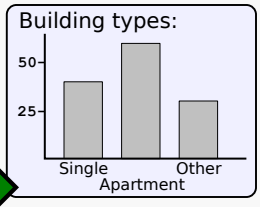
The specification of the metadata implementation of 3D Tiles 1.1 includes a JSON Schema defintion that describes a serialization format for the metadata. The metadata schema definitions can be shared within a tileset and among different tilesets, and be augmented with application-specific semantic data models.

5.1. Metadata Statistics

The metadata for a tileset can optionally contain **statistics** about all entities in a tileset, on a per-class basis. The statistics for numeric values as well as arrays, vectors, or matrices containing numeric values include common statistical measures, like the minimum- and maximum values, the mean or the standard deviation. For discrete types (enums or fixed-length arrays of enums), the statistics include the number of occurrences of each enum value.

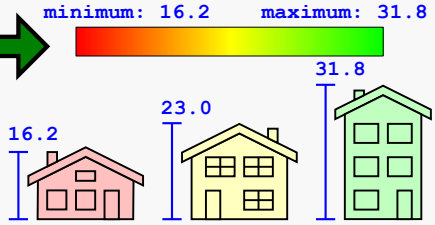
Statistics can be useful vor various analytical tasks and data visualization, but also for rendering and styling the rendered content based on the statistical information.

```
"statistics": {
  "classes": {
    "building": {
      "count": 130,
      "properties": {
        "buildingType": {
          "occurrences": {
            "Single": 40,
            "Apartment": 60,
            "Other": 30
          }
        },
        "height": {
          "minimum": 16.2,
          "maximum": 31.8
        }
      }
    }
  }
}
```



The number of occurrences can be used to visualize the distribution of different types with a bar chart. Here, the bar chart shows the different numbers of building types that appear in one tileset.

The statistics can also be useful for maping the metadata values that appear in one tileset to a certain color range:



In this example, the minimum and maximum heights of all buildings are used to map the building heights to a color range, and render the buildings with a color that depends on their height.

6. glTF Extensions for 3D Tiles 1.1

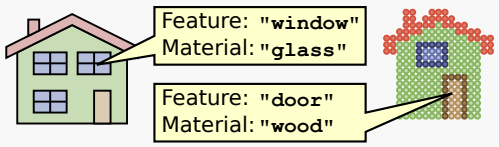
In 3D Tiles 1.0, the Batch IDs and Batch Tables offered mechanisms to identify individual features within the tile content, and to associate these features with metadata. With glTF becoming the primary tile format for 3D Tiles 1.1, these functionalities are emulated with the following glTF extensions.

6.1. The `EXT_mesh_features` glTF Extension

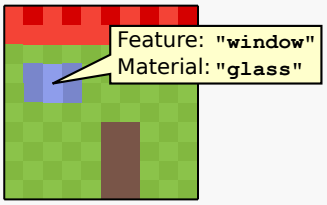
The **`EXT_mesh_features`** extension is an extension for glTF 2.0 that defines a means of assigning identifiers to geometry and subcomponents of geometry within a glTF asset. These subcomponents are referred to as **features**, and the identifiers for these features can be associated with a model in different ways:

- **By vertex**: The feature IDs are stored as standard glTF vertex attributes
- **By texture coordinates**: The feature IDs are stored in the channels of a standard glTF texture
- **By index**: The feature ID is assigned implicitly to vertices, by using the vertex index as an ID

A feature could be a single building in a 3D model of a city, or a part of a 3D model of a building. The feature is then defined by a subset of vertices of the 3D model, or a subset of points in a point cloud:



A feature can be certain region on the surface of a textured model:



6.2. The `EXT_structural_metadata` glTF Extension

The **`EXT_structural_metadata`** extension is an extension for glTF 2.0 that implements the 3D Metadata Specification. The structure of metadata is defined with a schema. This schema contains information about classes, their properties, and the types of these properties. An instance of such a metadata class can be created from a set of values that conforms to the properties of the class.

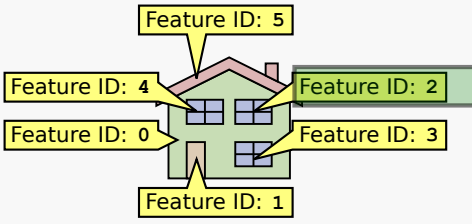
In addition to the JSON schema specification for defining the metadata structure, the extension defines methods for storing large amounts of metadata within a glTF asset, in fine-grained but compact binary form:

- **Property tables:** The values for each property are stored in a standard glTF buffer view. This creates a compact tabular representation of the metadata values.
- **Property Textures:** The property values are stored in the channels of a standard glTF texture. This way, high-frequency data can be associated with less detailed geometry surfaces.

6.3. Combining the glTF Extensions for 3D Tiles 1.1

The **`EXT_mesh_features`** and **`EXT_structural_metadata`** extension can be combined, and offer a mechanism for identifying features within an asset, and associating these features with metadata.

The **`EXT_mesh_features`** extension allows identifying features within an asset. Here, this is a model of a house where individual components receive a feature ID:



The **`EXT_structural_metadata`** extension allows defining the structure of metadata, and storing metadata values in a compact form. Here, a property table stores the metadata values. Each row stores the values for one feature. The columns correspond to the properties of a metadata class. The values are then used to create an instance of this metadata class:

Feature ID	Component	Material
0	Walls	Vinyl Panels
1	Door	Wood
2	Window	Glass
3	Window	Glass
4	Window	Glass
5	Roof	Shingles