3D Tiles Next is a set of extensions that are added to the 3D Tiles 1.0 specification. The new features that are offered with these extensions can be combined, so that they create a powerful toolset for the next generation of geospatial applications.

0. What's New in 3D Tiles Next

- **A set of new extensions for the 3D Tiles standard:**
  - **Simpler definition of tile content**: The `3DTILES_content_gltf` extension allows storing glTF assets directly as the tile content.
  - **More flexibility for organizing tile content**: With the `3DTILES_multiple_contents` extension, one tile can include the same content in different formats, or contain groups of content.
  - **Implicit tiling**: Tilesets consisting of quadtrees or octrees can be represented in a compact binary form using the `3DTILES_implicit_tiling` extension. The extension enables random access to tiles in the implicit hierarchy, and allows new, more efficient traversal algorithms.
  - **New bounding volume types**: Spatial indexing with S2 Geometry cells is supported with the `3DTILES_bounding_volume_s2` extension. The S2 Geometry bounding volumes are suitable for representing global-scale tilesets without singularities at the poles.
- **Versatile metadata support**:
  - **Metadata for tilesets, tiles and groups of content:** The `3DTILES_metadata` extension allows associating metadata with the core elements of a tileset
  - **Metadata for vertices, vertex groups and texels**: With the `EXT_mesh_features` extension for glTF 2.0, 3D Metadata can be stored even on the most fine-grained level of the 3D data: Individual vertices or texels may receive their own metadata information.

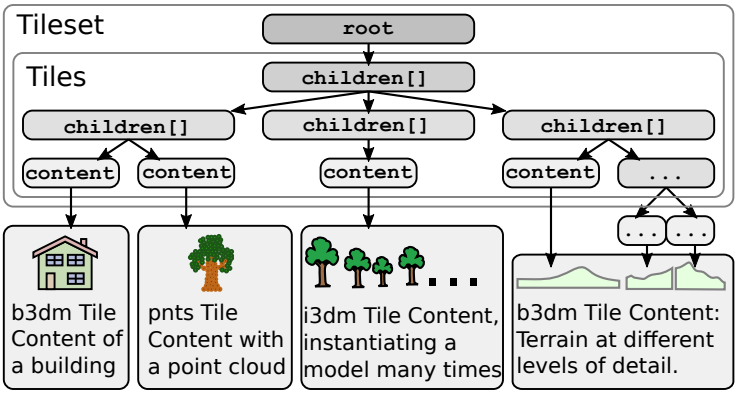
1. A Quick Introduction to 3D Tiles

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

3D Tiles 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



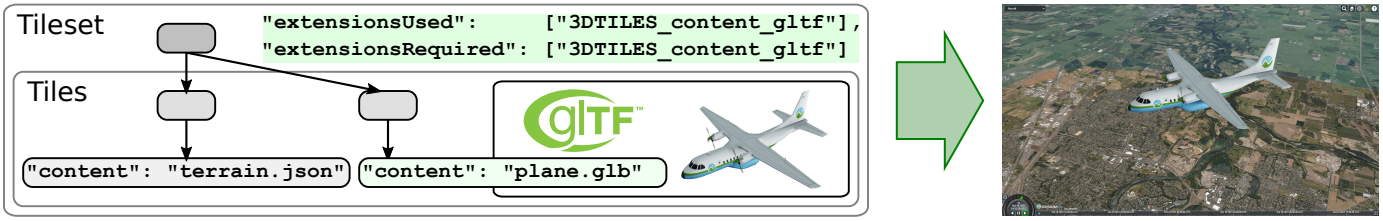
Each tile content can consist of multiple **features**. A single feature may be one part of a model in B3DM content, a point or a group of points in PNTS content, or an instance in I3DM content.

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 Extensions of 3D Tiles

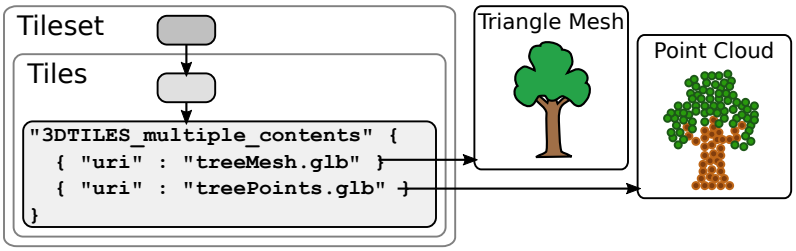
2.1. The `3DTILES_content_gltf` Extension

The **`3DTILES_content_gltf`** extension improves the interoperability between 3D Tiles and the glTF ecosystem. By declaring the extension in the tileset, glTF assets can be used as the content of any tile.



2.2. The `3DTILES_multiple_contents` Extension

With the **`3DTILES_multiple_contents`** extension, 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:

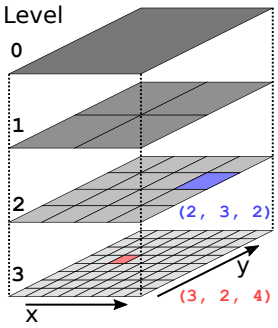


When combining this extension with the **`3DTILES_metadata`** extension, the contents can also be arranged in groups, and these groups can be associated with metadata. This allows applications to perform styling or optimizatons based on the group that the content belongs to.

2.3. The `3DTILES_implicit_tiling` Extension

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: The **`3DTILES_implicit_tiling`** extension defines 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.

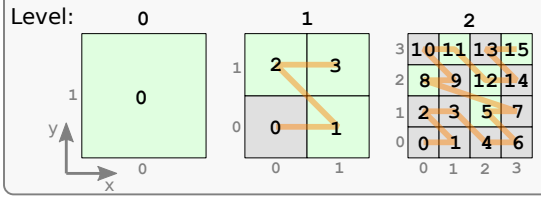
The extension supports different **subdivision schemes** for the implicit tile hierarchy: The bounding volume of the root tile is subdivided recursively into four or eight equal-sized parts, forming a **quadtree** or **octree**.



Within this implicit hierarchy, individual tiles can be accessed directly with their local **tile coordinates**: The coordinates of a tile within an octree 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:



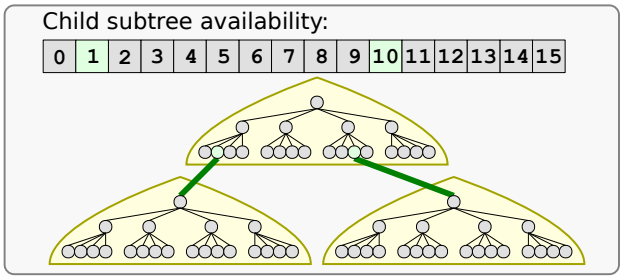
Level:	0	1			2															
Index in level:	0	0	1	2	3	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Index for data access:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

(2.3. The `3DTILES_implicit_tiling` Extension)

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.

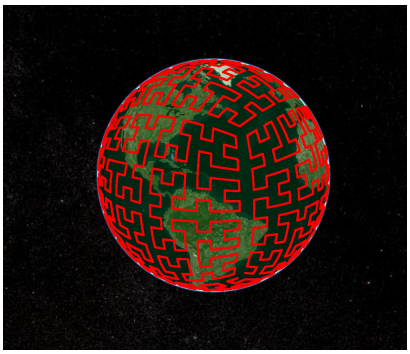
Together, this forms an implicit hierarchy by compositing a tree of subtrees. The entire tileset is described by this tree of subtrees.



2.4. The `3DTILES_bounding_volume_s2` Extension

Typically, Geographic Information Systems project map data on a plane. This process often causes distortions and singularities at the poles, making it difficult to accurately represent these regions with standard quadtree- or octree bounding volume hierarchies.

The **`3DTILES_bounding_volume_s2`** extension offers a new kind of bounding volume that does not suffer from these limitations. The bounding volumes are based on **S2 Geometry Cells**. These cells are created by projecting the six sides of a cube on the globe and hierarchically subdividing these cells. The cells on each subdivision level can then be accessed with a linear index, the S2 Cell ID. This index can be imagined as a single curve covering the whole surface of the globe.



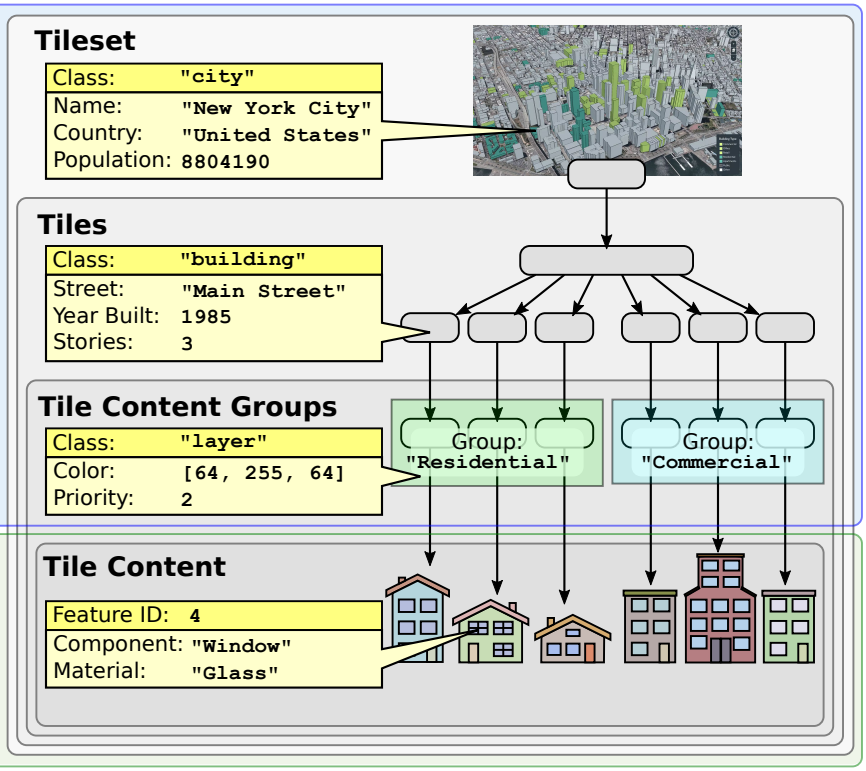
The actual structure of the hierarchy S2 Cell bounding volumes is still a quadtree or octree. This means that the **`3DTILES_bounding_volume_s2`** extension can be combined with the **`3DTILES_implicit_tiling`** extension to store the hierarchy structure in a compact binary form.

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. There are two implementations of this concept in the context of 3D Tiles Next that allow associating metadata with 3D data on all levels of granularity:

`3DTILES_metadata`

The **`3DTILES_metadata`** extension is an extension for 3D Tiles. It allows assigning metadata to the core elements of the 3D Tiles standard, namely tilesets, tiles, and groups of tile content.



`EXT_mesh_features`

The **`EXT_mesh_features`** extension is an extension for glTF 2.0. It allows assigning metadata to features in a model on the level of vertices or groups of vertices, or texels.

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.

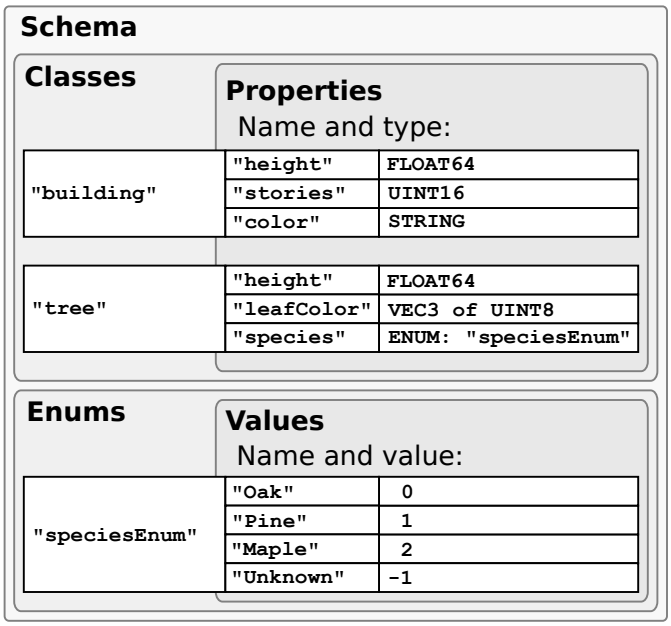
The primitive types include numeric types - namely, integer- and floating-point types with different sizes - as well boolean, enum types and strings. Fixed- and variable length array types are defined based on these primitive types. The type system also includes dedicated types for 2D, 3D, and 4D vectors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

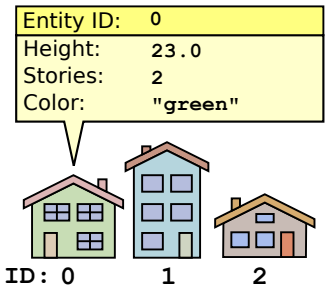
The schema itself only describes the structure of the metadata and the types of properties. But it does not define how the property values are stored. This allows reusing the schema definition across different assets and file formats, and combining it with different storage formats.

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:



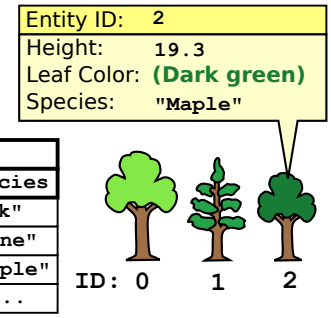
The columns and their types are determined by the schema:

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

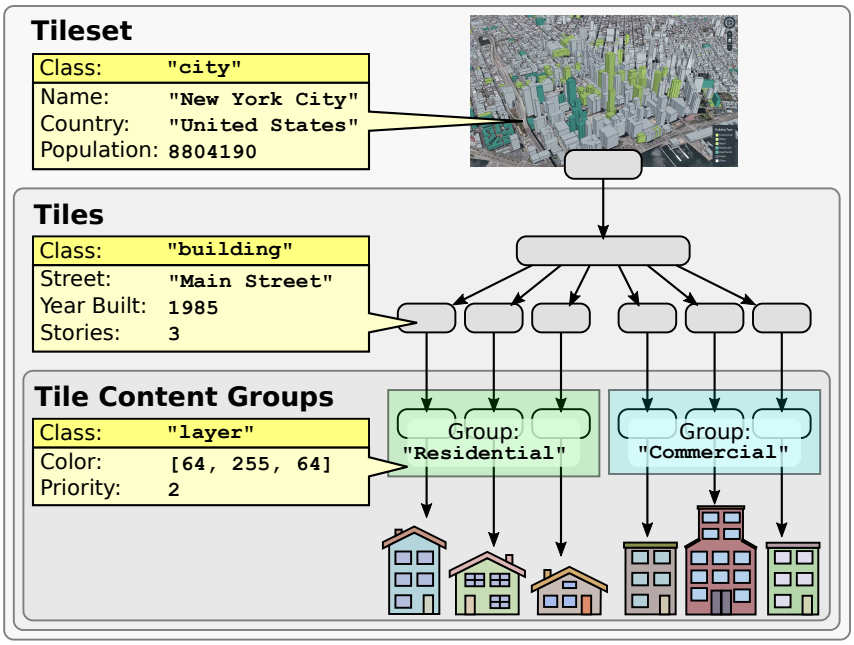
The specification contains further details about the exact encoding and serialization formats. Implementations of the 3D Metadata specification may define their own, custom serialization- and storage formats.

5. The `3DTILES_metadata` Extension

The **`3DTILES_metadata`** extension is an extension for 3D Tiles 1.0. It is an implementation of the 3D Metadata specification and allows assigning metadata to elements of a tileset.

By assigning metadata to tilesets, tiles, or groups of tiles, 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.



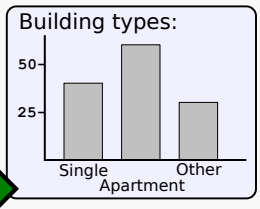
The specification of the **`3DTILES_metadata`** extension includes a JSON Schema defintion that describes a serialization format for the 3D Tiles Metadata. The metadata schema definitions can be shared within a tileset and among different tilesets, and be augmented with application-specific semantic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

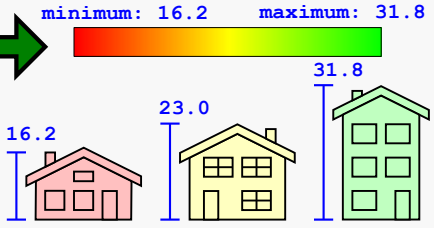
Statistics can be useful vor various analytical tasks and data visualization, but also for rendering and styling the rendered content based on the statistical information.

```
"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:

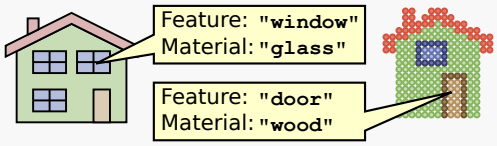


In this example, the minimum and maximum heights of all buildings are used to map the building heights to a color range, and render the buildings with a color that depends on their height.

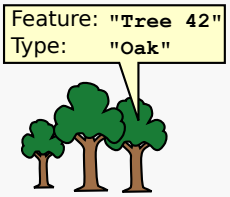
6. The `EXT_mesh_features` Extension

The **`EXT_mesh_features`** 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.
The extension allows assigning metadata to geometry and subcomponents of geometry that is stored in a glTF asset. These subcomponents are called **features**. Features can be defined on different levels of granularity, as shown with the following examples:

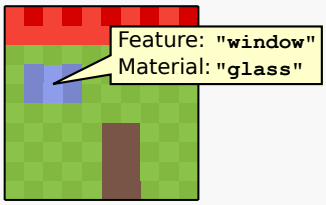
A feature could be a single building in a 3D model of a city, or a part of a 3D model of a building. The feature is then defined by a subset of vertices of the 3D model, or a subset of points in a point cloud:



A feature can be one of multiple instances of the same model:



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

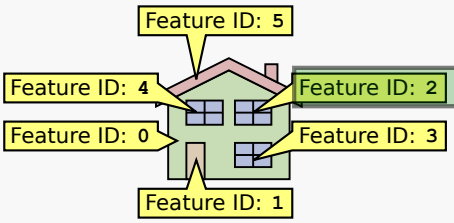


The features are connected to the actual metadata based on two concepts:

- **Identifying features**: Parts of the model are identified as features, and receive a unique **Feature ID**
- **Associating metadata with features**: The feature IDs are used to look up a set of metadata values that are then used to create an instance of a metadata class.

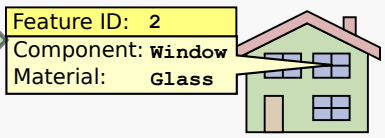
The following example shows how features are identified in a 3D model and associated with metadata values, to create an instance of a metadata class:

Identifying Features:
A model of a house where individual components receive a feature ID:



Associating Metadata with Features: A property table stores 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



The **`EXT_mesh_features`** extension defines different mechanisms for identifying features of a model:

- **By vertex**: Individual vertices or points of a point cloud can receive their own feature ID. These IDs may be assigned explicitly, as vertex attributes, or implicitly, by defining a pattern for the creation of the IDs. Groups of vertices or points can receive the same feature ID when they are part of one subcomponent that should be regarded as a single feature.
- **By texture coordinates:** Feature IDs can be stored in a texture and accessed using texture coordinates of the vertices. This allows regions on the surface of the geometry to be identified as features.
- **By GPU instance:** When multiple instances of the same mesh are rendered, then each instance can receive its own feature ID.

Additionally, the extension defines two methods of storing metadata values for the features in a fine-grained but compact binary form:

- **Property tables:** The values for each property are stored in a standard glTF buffer view, indexed by the feature ID. 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.