INTERNATIONAL STANDARD

ISO/IEC 12113

First edition
2022-07

Information technology — Runtime 3D asset delivery format — Khronos glTF™ 2.0

Technologies de l'information — Format de livraison d'actifs 3D d'exécution — Khronos glTF™ 2.0

Reference number
ISO/IEC 12113:2022(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see https://patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Khronos (as glTF™ 2.0 Specification) and drafted in accordance with its editorial rules. It was adopted, under the JTC 1 PAS procedure, by Joint Technical Committee ISO/IEC JTC 1, *Information technology*.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Table of Contents

Chapter 1. Foreword

Some parts of this Specification are non-normative through being explicitly identified as purely informative, and do not define requirements necessary for compliance and so are outside the Scope of this Specification.

Where this Specification includes normative references to external documents, only the specifically identified sections and functionality of those external documents are in Scope. Requirements defined by external documents not created by Khronos may contain contributions from non-members of Khronos not covered by the Khronos Intellectual Property Rights Policy.

Chapter 2. Introduction

2.1. General

This document, referred to as the “glTF Specification” or just the “Specification” hereafter, describes the glTF file format.

glTF is an API-neutral runtime asset delivery format. glTF bridges the gap between 3D content creation tools and modern graphics applications by providing an efficient, extensible, interoperable format for the transmission and loading of 3D content.

2.2. Document Conventions

The glTF Specification is intended for use by both implementers of the asset exporters or converters (e.g., digital content creation tools) and application developers seeking to import or load glTF assets, forming a basis for interoperability between these parties.

Specification text can address either party; typically, the intended audience can be inferred from context, though some sections are defined to address only one of these parties.

Any requirements, prohibitions, recommendations, or options defined by normative terminology are imposed only on the audience of that text.

2.2.1. Normative Terminology and References

The key words **MUST**, **MUST NOT**, **REQUIRED**, **SHALL**, **SHALL NOT**, **SHOULD**, **SHOULD NOT**, **RECOMMENDED**, **MAY**, and **OPTIONAL** in this document are to be interpreted as described in BCP 14.

These key words are highlighted in the specification for clarity.

References to external documents are considered normative if the Specification uses any of the normative terms defined in this section to refer to them or their requirements, either as a whole or in part.

2.2.2. Informative Language

Some language in the specification is purely informative, intended to give background or suggestions to implementers or developers.

If an entire chapter or section contains only informative language, its title is suffixed with “(Informative)”. If not designated as informative, all chapters, sections, and appendices in this document are normative.

All Notes, Implementation notes, and Examples are purely informative.

2.2.3. Technical Terminology

The glTF Specification makes use of linear algebra terms such as **axis**, **matrix**, **vector**, etc. to identify certain math constructs and their behaviors as defined in the International Electrotechnical Vocabulary.

The glTF Specification makes use of common engineering and graphics terms such as **image**, **buffer**, **texture**, etc. to identify and describe certain *glTF* constructs and their attributes, states, and behaviors. This section defines the basic meanings of these terms in the context of the Specification. The Specification text provides fuller definitions of the terms and elaborates, extends, or clarifies the definitions. When a term defined in this section is used in normative language within the Specification, the definitions within the Specification govern and supersede any meanings the terms may have in other technical contexts (i.e. outside the Specification).

accessor

An object describing the number and the format of data elements stored in a binary buffer.

animation

An object describing the keyframe data, including timestamps, and the target property affected by it.

back-facing

See facingness.

buffer

An external or embedded resource that represents a linear array of bytes.

buffer view

An object that represents a range of a specific buffer, and optional metadata that controls how the buffer's content is interpreted.

camera

An object defining the projection parameters that are used to render a scene.

facingness

A classification of a triangle as either front-facing or back-facing, depending on the orientation (winding order) of its vertices.

front-facing

See facingness.

image

A two dimensional array of pixels encoded as a standardized bitstream, such as PNG.

indexed geometry

A mesh primitive that uses a separate source of data (index values) to assemble the primitive's topology.

linear blend skinning

A skinning method that computes a per-vertex transformation matrix as a linear weighted sum of transformation matrices of the designated nodes.

material

A parametrized approximation of visual properties of the real-world object being represented by a mesh primitive.

mesh

A collection of mesh primitives.

mesh primitive

An object binding indexed or non-indexed geometry with a material.

mipmap

A set of image representations consecutively reduced by the factor of 2 in each dimension.

morph target

An altered state of a mesh primitive defined as a set of difference values for its vertex attributes.

node

An object defining the hierarchy relations and the local transform of its content.

non-indexed geometry

A mesh primitive that uses linear order of vertex attribute values to assemble the primitive's topology.

normal

A unit XYZ vector defining the perpendicular to the surface.

root node

A node that is not a child of any other node.

sampler

An object that controls how image data is sampled.

scene

An object containing a list of root nodes to render.

skinning

The process of computing and applying individual transforms for each vertex of a mesh primitive.

tangent

A unit XYZ vector defining a tangential direction on the surface.

texture

An object that combines an image and its sampler.

topology type

State that controls how vertices are assembled, e.g. as lists of triangles, strips of lines, etc.

vertex attribute

A property associated with a vertex.

winding order

The relative order in which vertices are defined within a triangle

wrapping

A process of selecting an image pixel based on normalized texture coordinates.

2.2.4. Normative References

The following documents are referenced by normative sections of the specification.

2.2.4.1. External Specifications

Bradner, S., *Key words for use in RFCs to Indicate Requirement Levels*, BCP 14, RFC 2119, March 1997. Leiba, B., *Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words*, BCP 14, RFC 8174, May 2017. https://www.rfc-editor.org/info/bcp14

IEC 60050-102 *International Electrotechnical Vocabulary (IEV) - Part 102: Mathematics - General concepts and linear algebra* https://webstore.iec.ch/publication/160

IEC 60050-103 *International Electrotechnical Vocabulary (IEV) - Part 103: Mathematics - Functions* https://webstore.iec.ch/publication/161

Note

An online version of these standards is available at https://www.electropedia.org/

The Unicode Consortium, *The Unicode Standard* https://www.unicode.org/versions/latest/

Bray, T., Ed., *The JavaScript Object Notation (JSON) Data Interchange Format*, STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, https://www.rfc-editor.org/info/rfc8259

ISO/IEC 60559 *Floating-point arithmetic* https://www.iso.org/standard/80985.html

ISO/IEC 15948 *Portable Network Graphics (PNG): Functional specification* https://www.iso.org/standard/29581.html

Note

A free version of this standard is available from W3C: https://www.w3.org/TR/PNG/

ISO/IEC 10918-1 *Digital compression and coding of continuous-tone still images: Requirements and guidelines* https://www.iso.org/standard/18902.html

Note

An earlier edition of this standard called ITU Recommendation T.81 is available from W3C: https://www.w3.org/Graphics/JPEG/itu-t81.pdf

ISO/IEC 10918-5 *Digital compression and coding of continuous-tone still images: JPEG File Interchange Format (JFIF)* https://www.iso.org/standard/54989.html

Note

An earlier edition of this standard is available from W3C: https://www.w3.org/Graphics/JPEG/jfif3.pdf

CIPA DC-008-Translation-2019 *Exchangeable image file format for digital still cameras* https://www.cipa.jp/std/documents/download_e.html?DC-008-Translation-2019-E

Masinter, L., *The "data" URL scheme*, RFC 2397, DOI 10.17487/RFC2397, August 1998, https://www.rfc-editor.org/info/rfc2397

Berners-Lee, T., Fielding, R., and L. Masinter, *Uniform Resource Identifier (URI): Generic Syntax*, STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, https://www.rfc-editor.org/info/rfc3986

Duerst, M. and M. Suignard, *Internationalized Resource Identifiers (IRIs)*, RFC 3987, DOI 10.17487/RFC3987, January 2005, https://www.rfc-editor.org/info/rfc3987

Fielding, R., Ed., and J. Reschke, Ed., *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*, RFC 7230, DOI 10.17487/RFC7230, June 2014, https://www.rfc-editor.org/info/rfc7230

IEC 61966-2-1 *Default RGB colour space - sRGB* https://webstore.iec.ch/publication/6169

Note

The encoding characteristics of sRGB are freely available from ICC: https://www.color.org/chardata/rgb/srgb.xalter

Recommendation ITU-R BT.709-6 *Parameter values for the HDTV standards for production and international programme exchange* https://www.itu.int/rec/R-REC-BT.709-6-201506-I

MikkTSpace https://github.com/mmikk/MikkTSpace

Thomas Porter and Tom Duff. 1984. *Compositing digital images.* SIGGRAPH Comput. Graph. 18, 3 (July 1984), 253–259. DOI: https://doi.org/10.1145/964965.808606

Note

A free version of this paper is available from Pixar: https://graphics.pixar.com/library/Compositing/

2.2.4.2. Media Type Registrations

IANA. *model/gltf+json Media Type*. https://www.iana.org/assignments/media-types/model/gltf+json

IANA. *model/gltf-binary Media Type*. https://www.iana.org/assignments/media-types/model/gltf-binary

IANA. *application/gltf-buffer Media Type*. https://www.iana.org/assignments/media-types/application/gltf-buffer

IANA. *application/octet-stream Media Type*. https://www.iana.org/assignments/media-types/application/octet-stream

Freed, N. and N. Borenstein, *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*, RFC 2046, DOI 10.17487/RFC2046, November 1996, https://www.rfc-editor.org/info/rfc2046

IANA. *image/png Media Type*. https://www.iana.org/assignments/media-types/image/png

2.3. Motivation and Design Goals (Informative)

glTF is an open interoperable 3D asset 'transmission' format that is compact, and efficient to process and render at runtime. glTF 2.0 is designed to be vendor- and runtime-neutral, usable by a wide variety of native and web-based engines and applications regardless of underlying platforms and 3D graphics APIs.

glTF's focus on run-time efficiency is a different design goal than typical 3D 'authoring' formats. Authoring formats are typically more verbose, with higher processing overheads, to carry authoring data that is no longer needed after iterative design is complete. glTF is complementary to authoring formats, providing a common, interoperable distillation target for publishing 3D assets to a wide audience of end users.

A primary goal of glTF is to be deployable on a wide range of devices and platforms, including the web and mobile devices with limited processing and memory resources. glTF can be evolved, to keep pace with growing compute capabilities over time. This helps to foster broad industry consensus on 3D functionality that can be used ubiquitously, including Physically Based Rendering.

glTF combines an easily parsable JSON scene description with one or more binary resources representing geometry, animations, and other rich data. These binary resources can often be loaded directly into GPU buffers with no additional parsing or processing, combining the faithful preservation of full hierarchical scenes, nodes, meshes, cameras, materials, and animations with efficient delivery and fast loading.

glTF has been designed to meet the following goals:

- **Compact file sizes.** The plain text glTF JSON file description is compact and rapid to parse. All large data such as geometry, textures and animations are stored in binary files that are significantly smaller than equivalent text representations.
- **Runtime-independence.** glTF is purely an asset format and does not mandate any runtime behavior. This enables its use by any application for any purpose, including display using any rendering technology, up to and including path tracing renderers.
- **Complete 3D scene representation.** Not restricted to single objects, glTF can represent entire scenes, including nodes, transformations, transform hierarchy, meshes, materials, cameras, and animations.
- **Extensibility.** glTF is fully extensible, enabling the addition of both general-purpose and vendor-specific extensions, including geometry and texture compression. Widely adopted extensions may be considered for integration into future versions of the glTF specification.

The following are outside the scope of glTF 2.0:

- **glTF is not a streaming format.** The binary data in glTF is inherently streamable, and the buffer design allows for fetching data incrementally, but there are no other streaming constructs in glTF 2.0.
- **glTF is not an authoring format.** glTF deliberately does not retain 3D authoring information, in order to preserve runtime efficiency, however glTF files may be ingested by 3D authoring tools for remixing.
- **glTF is not intended to be human-readable,** though by virtue of being represented in JSON, it is developer-friendly.

2.4. glTF Basics

A glTF asset is represented by:

- A JSON-formatted file (`.gltf`) containing a full scene description: node hierarchy, materials, cameras, as well as descriptor information for meshes, animations, and other constructs.
- Binary files (`.bin`) containing geometry, animation, and other buffer-based data.
- Image files (`.jpg`, `.png`) containing texture images.

Binary and image resources **MAY** also be embedded directly in JSON using Data URI or stored side-by-side with JSON in GLB container.

A valid glTF asset **MUST** specify its version.

2.5. Versioning

Any updates made to the glTF Specification in a minor version **MUST** be backward and forward compatible. Backward compatibility means that any client implementation that supports loading a glTF 2.x asset will also be able to load a glTF 2.0 asset. Forward compatibility means that a client implementation that only supports glTF 2.0 can load glTF 2.x assets while gracefully ignoring any new features it does not understand.

A minor version update **MAY** introduce new features but **MUST NOT** change any previously existing behavior. Existing functionality **MAY** be deprecated in a minor version update, but it **MUST NOT** be removed.

Major version updates **MAY** be incompatible with previous versions.

2.6. File Extensions and Media Types

- JSON glTF files **SHOULD** use `.gltf` extension and `model/gltf+json` Media Type.
- glTF files stored in GLB container **SHOULD** use `.glb` extension and `model/gltf-binary` Media Type.
- Files representing binary buffers **SHOULD** use either:

  - `.bin` file extension with `application/octet-stream` Media Type;
  - `.bin`, `.glbin`, or `.glbuf` file extensions with `application/gltf-buffer` Media Type.
- PNG images **SHOULD** use `.png` file extension with `image/png` Media Type;
  - PNG images **SHOULD NOT** contain animations, non-square pixel ratios, or embedded ICC profiles. Such features, if present, **MUST** be ignored by client implementations.
- JPEG images **SHOULD** use `.jpeg` or `.jpg` file extensions with `image/jpeg` Media Type
  - JPEG images **MUST** be compatible with JPEG File Interchange Format.
  - JPEG images **SHOULD NOT** contain embedded ICC profiles. If present, embedded ICC profiles **MUST** be ignored by client implementations.
  - Exchangeable image file format (Exif) chunks **MAY** be ignored by client implementations.

Implementation Note

Certain Exif chunks, e.g., “Orientation”, may severely impact an asset’s portability.

2.7. JSON Encoding

Although glTF Specification does not define any subset of the JSON format, implementations **SHOULD** be aware of its peculiar properties that could affect asset interoperability.

1. glTF JSON data **SHOULD** be written with UTF-8 encoding without BOM. This requirement is not applied when a glTF implementation does not control string encoding. glTF implementations **SHOULD** adhere to RFC 8259, Section 8.1, with regards to treating BOM presence.
2. ASCII characters stored in glTF JSON **SHOULD** be written without JSON escaping.

Example

`"buffer"` instead of `"\u0062\u0075\u0066\u0066\u0065\u0072"`.

3. Non-ASCII characters stored in glTF JSON **MAY** be escaped.

Example

These two examples represent the same glTF JSON data.

```
{
    "asset": {
        "version": "2.0"
    },
    "nodes": [
        {
            "name": "ky6"
        },
        {
            "name": "□□□"
        }
    ]
}
```

```
{
    "asset": {
        "version": "2.0"
    },
    "nodes": [
        {
            "name": "\u043a\u0443\u0431"
        },
        {
            "name": "\u7acb\u65b9\u9ad4"
        }
    ]
}
```

4. Property names (keys) within JSON objects **SHOULD** be unique. glTF client implementations **SHOULD** override lexically preceding values for the same key.
5. Some of glTF properties are defined as integers in the schema. Such values **MAY** be stored as decimals with a zero fractional part or by using exponent notation. Regardless of encoding, such properties **MUST NOT** contain any non-zero fractional value.

Example

`100`, `100.0`, and `1e2` represent the same value. See RFC 8259, Section 6 for more details.

6. Non-integer numbers **SHOULD** be written in a way that preserves original values when these numbers are read back, i.e., they **SHOULD NOT** be altered by JSON serialization / deserialization roundtrip.

Implementation Note

This is typically achieved with algorithms like Grisu2 used by common JSON libraries.

2.8. URIs

glTF assets use URIs or IRIs to reference buffers and image resources. Assets **MAY** contain at least these two URI types:

- **Data URIs** that embed binary resources in the glTF JSON as defined by the RFC 2397. The Data URI's `mediatype` field **MUST** match the encoded content.

Implementation Note

Base64 encoding used in Data URI increases the payload's byte length by 33%.

- **Relative paths** — `path-noscheme` or `ipath-noscheme` as defined by RFC 3986, Section 4.2 or RFC 3987, Section 2.2 — without scheme, authority, or parameters. Reserved characters (as defined by RFC 3986, Section 2.2. and RFC 3987, Section 2.2.) **MUST** be percent-encoded.

Paths with non-ASCII characters **MAY** be written as-is, with JSON string escaping, or with percent-encoding; all these options are valid. For example, the following three paths point to the same resource:

```
{
    "images": [
        {
            "uri": "grande_sphère.png"
        },
        {
            "uri": "grande_sph\u00E8re.png"
        },
        {
            "uri": "grande_sph%C3%A8re.png"
        }
    ]
}
```

Client implementations **MAY** optionally support additional URI components. For example `http://` or `file://` schemes, authorities, hostnames, absolute paths, and query or fragment parameters. Assets containing these additional URI components would be less portable.

Implementation Note

This allows the application to decide the best approach for delivery: if different assets share many of the same geometries, animations, or textures, separate files may be preferred to reduce the total amount of data requested. With separate files, applications may progressively load data and do not need to load data for parts of a model that are not visible. If an application cares more about single-file deployment, embedding data may be preferred even though it increases the overall size due to base64 encoding and does not support progressive or on-demand loading. Alternatively, an asset could use the GLB container to store JSON and binary data in one file without base64 encoding. See GLB File Format Specification for details.

URIs **SHOULD** undergo syntax-based normalization as defined by RFC 3986, Section 6.2.2, RFC 3987, Section 5.3.2, and applicable schema rules (e.g., RFC 7230, Section 2.7.3 for HTTP) on export and/or import.

Implementation Note

While the specification does not explicitly disallow non-normalized URIs, their use may be unsupported or lead to unwanted side-effects — such as security warnings or cache misses — on some platforms.

Chapter 3. Concepts

3.1. General

The figure below shows relations between top-level arrays in a glTF asset. See the Properties Reference.

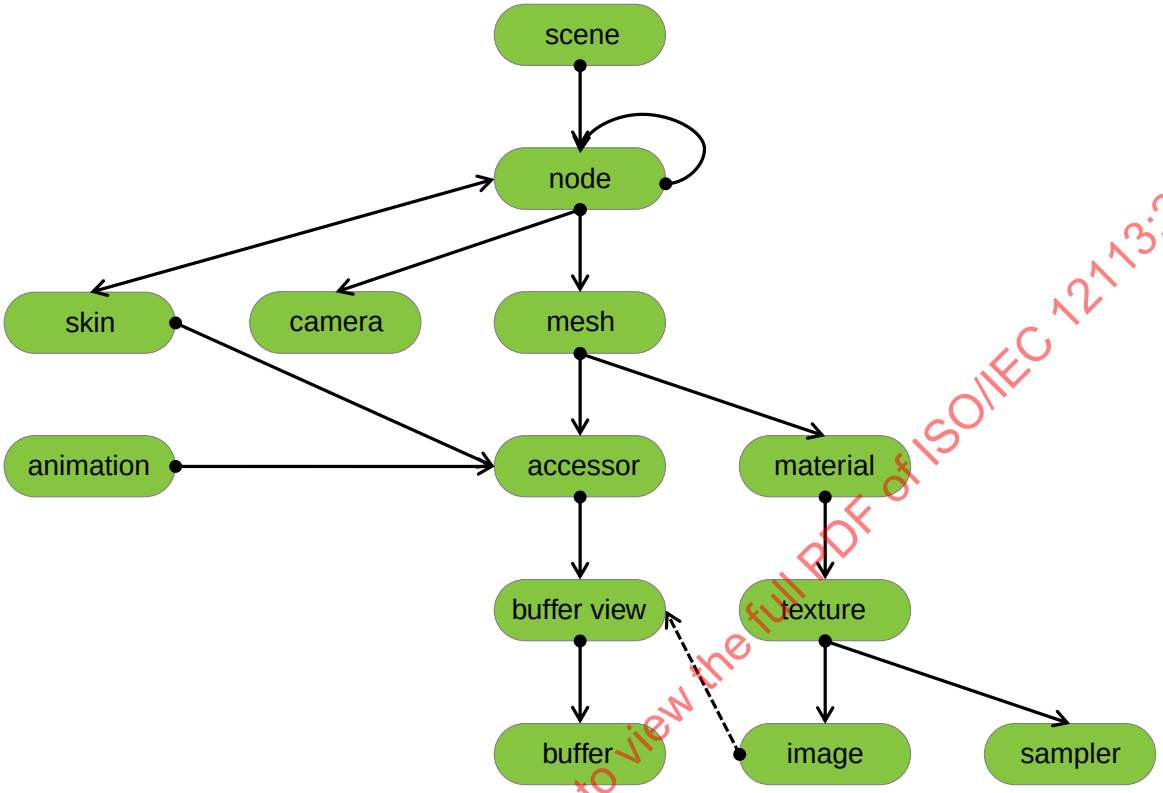

Figure 1. glTF Object Hierarchy

3.2. Asset

Each glTF asset **MUST** have an `asset` property. The `asset` object **MUST** contain a `version` property that specifies the target glTF version of the asset. Additionally, an optional `minVersion` property **MAY** be used to specify the minimum glTF version support required to load the asset. The `minVersion` property allows asset creators to specify a minimum version that a client implementation **MUST** support in order to load the asset. This is very similar to the `extensionsRequired` concept described in Section 3.12, where an asset **SHOULD NOT** be loaded if the client does not support the specified extension. Additional metadata **MAY** be stored in optional properties such as `generator` or `copyright`. For example,

```
{
    "asset": {
        "version": "2.0",
        "generator": "collada2gltf@f356b99aef8868f74877c7ca545f2cd206b9d3b7",
        "copyright": "2017 (c) Khronos Group"
    }
}
```

Implementation Note

Client implementations should first check whether a `minVersion` property is specified and ensure both major and minor versions can be supported. If no `minVersion` is specified, then clients should check the `version` property and ensure the major version is supported. Clients that load GLB format should also check for the `minVersion` and `version` properties in the JSON chunk as the version specified in the GLB header only refers to the GLB container version.

3.3. Indices and Names

Entities of a glTF asset are referenced by their indices in corresponding arrays, e.g., a `bufferView` refers to a `buffer` by specifying the buffer's index in `buffers` array. For example:

```
{
    "buffers": [
        {
            "byteLength": 1024,
            "uri": "path-to.bin"
        }
    ],
    "bufferViews": [
        {
            "buffer": 0,
            "byteLength": 512,
            "byteOffset": 0
        }
    ]
}
```


In this example, `buffers` and `bufferViews` arrays have only one element each. The bufferView refers to the buffer using the buffer's index: `"buffer": 0`.

Indices **MUST** be non-negative integer numbers. Indices **MUST** always point to existing elements.

Whereas indices are used for internal glTF references, optional *names* are used for application-specific uses such as display. Any top-level glTF object **MAY** have a `name` string property for this purpose. These property values are not guaranteed to be unique as they are intended to contain values created when the asset was authored.

For property names, glTF usually uses camel case, `likeThis`.

3.4. Coordinate System and Units

glTF uses a right-handed coordinate system. glTF defines +Y as up, +Z as forward, and -X as right; the front of a glTF asset faces +Z.

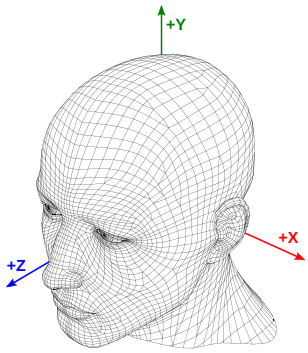

Figure 2. glTF Coordinate System Orientation

The units for all linear distances are meters.

All angles are in radians.

Positive rotation is counterclockwise.

Red, Green, and Blue primary colors use Recommendation ITU-R BT.709 chromaticity coordinates.

> *Implementation Note*
>
> Chromaticity coordinates define the interpretation of each primary color channel of the color model. In the context of a typical display, color primaries describe the color of the red, green, and blue phosphors or filters. Unless a wide color gamut output is explicitly used, client implementations usually do not need to convert colors. Future specification versions or extensions may allow other color primaries (such as P3).

3.5. Scenes

3.5.1. Overview

glTF 2.0 assets **MAY** contain zero or more *scenes*, the set of visual objects to render. Scenes are defined in a `scenes` array. All nodes listed in `scene.nodes` array **MUST** be root nodes, i.e., they **MUST NOT** be listed in a `node.children` array of any node. The same root node **MAY** appear in multiple scenes.

An additional root-level property, `scene` (note singular), identifies which of the scenes in the array **SHOULD** be displayed at load time. When `scene` is undefined, client implementations **MAY** delay rendering until a particular scene is requested.

A glTF asset that does not contain any scenes **SHOULD** be treated as a library of individual entities

such as materials or meshes.

The following example defines a glTF asset with a single scene that contains a single node.

```
{
    "nodes": [
        {
            "name": "singleNode"
        }
    ],
    "scenes": [
        {
            "name": "singleScene",
            "nodes": [
                0
            ]
        }
    ],
    "scene": 0
}
```

3.5.2. Nodes and Hierarchy

glTF assets **MAY** define *nodes*, that is, the objects comprising the scene to render.

Nodes **MAY** have transform properties, as described later.

Nodes are organized in a parent-child hierarchy known informally as the *node hierarchy*. A node is called a *root node* when it doesn't have a parent.

The node hierarchy **MUST** be a set of disjoint strict trees. That is node hierarchy **MUST NOT** contain cycles and each node **MUST** have zero or one parent node.

The node hierarchy is defined using a node's `children` property, as in the following example:

```
{
    "nodes": [
        {
            "name": "Car",
            "children": [1, 2, 3, 4]
        },
        {
            "name": "wheel_1"
        },
        {
            "name": "wheel_2"
        },
        {
            "name": "wheel_3"
        },
        {
            "name": "wheel_4"
        }
    ]
}
```

The node named `Car` has four children. Each of those nodes could in turn have its own children, creating a hierarchy of nodes.

3.5.3. Transformations

Any node **MAY** define a local space transform either by supplying a `matrix` property, or any of `translation`, `rotation`, and `scale` properties (also known as *TRS properties*). `translation` and `scale` are 3D vectors in the local coordinate system. `rotation` is a unit quaternion value, XYZW, in the local coordinate system, where W is the scalar.

When `matrix` is defined, it **MUST** be decomposable to TRS properties.

Implementation Note

Transformation matrices cannot skew or shear.

When a node is targeted for animation (referenced by an `animation.channel.target`), only TRS properties **MAY** be present; `matrix` **MUST NOT** be present.

To compose the local transformation matrix, TRS properties **MUST** be converted to matrices and postmultiplied in the `T * R * S` order; first the scale is applied to the vertices, then the rotation, and then the translation.

Implementation Note

Non-invertible transforms (e.g., scaling one axis to zero) could lead to lighting and/or visibility artifacts.

Implementation Note

When the scale is zero on all three axes (by node transform or by animated scale), implementations are free to optimize away rendering of the node's mesh, and all of the node's children's meshes. This provides a mechanism to animate visibility. Skinned meshes must not use this optimization unless all of the joints in the skin are scaled to zero simultaneously.

The global transformation matrix of a node is the product of the global transformation matrix of its parent node and its own local transformation matrix. When the node has no parent node, its global transformation matrix is identical to its local transformation matrix.

In the example below, a node named `Box` defines non-default rotation and translation.

```
{
    "nodes": [
        {
            "name": "Box",
            "rotation": [
                0,
                0,
                0,
                1
            ],
            "scale": [
                1,
                1,
                1
            ],
            "translation": [
                -17.7082,
                -11.4156,
                2.0922
            ]
        }
    ]
}
```


The next example defines the transform for a node with attached camera using the `matrix` property rather than using the individual TRS values:

```
{
    "nodes": [
        {
            "name": "node-camera",
            "camera": 1,
            "matrix": [
                -0.99975,
                -0.00679829,
                0.0213218,
                0,
                0.00167596,
                0.927325,
                0.374254,
                0,
                -0.0223165,
                0.374196,
                -0.927081,
                0,
                -0.0115543,
                0.194711,
                -0.478297,
                1
            ]
        }
    ]
}
```


3.6. Binary Data Storage

3.6.1. Buffers and Buffer Views

3.6.1.1. Overview

A **buffer** is arbitrary data stored as a binary blob. The buffer **MAY** contain any combination of geometry, animation, skins, and images.

Binary blobs allow efficient creation of GPU buffers and textures since they require no additional parsing, except perhaps decompression.

glTF assets **MAY** have any number of buffer resources. Buffers are defined in the asset's `buffers` array.

While there's no hard upper limit on buffer's size, glTF assets **SHOULD NOT** use buffers bigger than 2^{53} bytes because some JSON parsers may be unable to parse their `byteLength` correctly. Buffers stored as GLB binary chunk have an implicit limit of 2^{32}-1 bytes.

All buffer data defined in this specification (i.e., geometry attributes, geometry indices, sparse accessor data, animation inputs and outputs, inverse bind matrices) **MUST** use little endian byte order.

The following example defines a buffer. The `byteLength` property specifies the size of the buffer file. The `uri` property is the URI to the buffer data.

```
{
   "buffers": [
       {
           "byteLength": 102040,
           "uri": "duck.bin"
       }
   ]
}
```

The byte length of the referenced resource **MUST** be greater than or equal to the `buffer.byteLength` property.

Buffer data **MAY** alternatively be embedded in the glTF file via `data:` URI with base64 encoding. When `data:` URI is used for buffer storage, its `mediatype` field **MUST** be set to `application/octet-stream` or `application/gltf-buffer`.

A *buffer view* represents a contiguous segment of data in a buffer, defined by a byte offset into the buffer specified in the `byteOffset` property and a total byte length specified by the `byteLength` property of the buffer view.

Buffer views used for images, vertex indices, vertex attributes, or inverse bind matrices **MUST** contain only one kind of data, i.e., the same buffer view **MUST NOT** be used both for vertex indices and vertex attributes.

When a buffer view is used by vertex indices or attribute accessors it **SHOULD** specify `bufferView.target` with a value of *element array buffer* or *array buffer* respectively.

Implementation Note

This allows client implementations to early designate each buffer view to a proper processing step, e.g, buffer views with vertex indices and attributes would be copied to the appropriate GPU buffers, while buffer views with image data would be passed to format-specific image decoders.

The `bufferView.target` value uses integer enums defined in the Properties Reference.

The following example defines two buffer views: the first holds the indices for an indexed triangle set, and the second holds the vertex data for the triangle set.

```
{
    "bufferViews": [
        {
            "buffer": 0,
            "byteLength": 25272,
            "byteOffset": 0,
            "target": 34963
        },
        {
            "buffer": 0,
            "byteLength": 76768,
            "byteOffset": 25272,
            "byteStride": 32,
            "target": 34962
        }
    ]
}
```

When a buffer view is used for vertex attribute data, it **MAY** have a `byteStride` property. This property defines the stride in bytes between each vertex. Buffer views with other types of data **MUST NOT** not define `byteStride` (unless such layout is explicitly enabled by an extension).

Buffers and buffer views do not contain type information. They simply define the raw data for retrieval from the file. Objects within the glTF asset (meshes, skins, animations) access buffers or buffer views via **accessors**.

3.6.1.2. GLB-stored Buffer

The glTF asset **MAY** use the GLB file container to pack glTF JSON and one glTF buffer into one file. Data for such a buffer is provided via the GLB-stored `BIN` chunk.

A buffer with data provided by the GLB-stored `BIN` chunk, **MUST** be the first element of `buffers` array and it **MUST** have its `buffer.uri` property undefined. When such a buffer exists, a `BIN` chunk **MUST** be present.

Any glTF buffer with undefined `buffer.uri` property that is not the first element of `buffers` array does not refer to the GLB-stored BIN chunk, and the behavior of such buffers is left undefined to accommodate future extensions and specification versions.

The byte length of the `BIN` chunk **MAY** be up to 3 bytes bigger than JSON-defined `buffer.byteLength` value to satisfy GLB padding requirements.

Implementation Note

Not requiring strict equality of chunk's and buffer's lengths slightly simplifies glTF to GLB conversion: `buffer.byteLength` does not need to be updated after applying GLB padding.

In the following example, the first buffer object refers to GLB-stored data, while the second points to external resource:

```
{
    "buffers": [
        {
            "byteLength": 35884
        },
        {
            "byteLength": 504,
            "uri": "external.bin"
        }
  ]
}
```

See GLB File Format Specification for details on GLB File Format.

3.6.2. Accessors

3.6.2.1. Overview

All binary data for meshes, skins, and animations is stored in buffers and retrieved via accessors.

An *accessor* defines a method for retrieving data as typed arrays from within a buffer view. The accessor specifies a component type (e.g., *float*) and a data type (e.g., `VEC3` for 3D vectors), which when combined define the complete data type for each data element. The number of elements is specified using the `count` property. Elements could be, e.g., vertex indices, vertex attributes, animation keyframes, etc.

The `byteOffset` property specifies the location of the first data element within the referenced buffer view. If the accessor is used for vertex attributes (i.e., it is referenced by a mesh primitive or its morph targets), the locations of the subsequent data elements are controlled by the `bufferView.byteStride` property. If the accessor is used for any other kind of data (vertex indices, animation keyframes, etc.), its data elements are tightly packed.

All accessors are stored in the asset's `accessors` array.

The following example shows two accessors, the first is a scalar accessor for retrieving a primitive's indices, and the second is a 3-float-component vector accessor for retrieving the primitive's position data.

```
{
    "accessors": [
        {
            "bufferView": 0,
            "byteOffset": 0,
            "componentType": 5123,
            "count": 12636,
            "max": [
                4212
            ],
            "min": [
                0
            ],
            "type": "SCALAR"
        },
        {
            "bufferView": 1,
            "byteOffset": 0,
            "componentType": 5126,
            "count": 2399,
            "max": [
                0.961799,
                1.6397,
                0.539252
            ],
            "min": [
                -0.692985,
                0.0992937,
                -0.613282
            ],
            "type": "VEC3"
        }
    ]
}
```


3.6.2.2. Accessor Data Types

componentType	Data Type	Signed	Bits
5120	*signed byte*	Signed, two's complement	8
5121	*unsigned byte*	Unsigned	8
5122	*signed short*	Signed, two's complement	16
5123	*unsigned short*	Unsigned	16
5125	*unsigned int*	Unsigned	32
5126	*float*	Signed	32

Signed 32-bit integer components are not supported.

Floating-point data **MUST** use IEEE-754 single precision format.

Values of `NaN`, `+Infinity`, and `-Infinity` **MUST NOT** be present.

`type`	Number of components
`"SCALAR"`	1
`"VEC2"`	2
`"VEC3"`	3
`"VEC4"`	4
`"MAT2"`	4
`"MAT3"`	9
`"MAT4"`	16

Element size, in bytes, is `(size in bytes of the 'componentType') * (number of components defined by 'type')`.

For example:

```
{
    "accessors": [
        {
            "bufferView": 1,
            "byteOffset": 7032,
            "componentType": 5126,
            "count": 585,
            "type": "VEC3"
        }
    ]
}
```

In this accessor, the `componentType` is `5126` (*float*), so each component is four bytes. The `type` is `"VEC3"`, so there are three components. The size of each element is 12 bytes (`4 * 3`). Thus, the accessor takes 7020 bytes (`[7032 ⋯ 14051]` inclusive range of the buffer view).

3.6.2.3. Sparse Accessors

Sparse encoding of arrays is often more memory-efficient than dense encoding when describing incremental changes with respect to a reference array. This is often the case when encoding morph targets (it is, in general, more efficient to describe a few displaced vertices in a morph target than transmitting all morph target vertices).

Similar to a standard accessor, a sparse accessor initializes an array of typed elements from data stored in a `bufferView`. When `accessor.bufferView` is undefined, the sparse accessor is initialized as an array of zeros of size `(size of the accessor element) * (accessor.count)` bytes.

On top of that, a sparse accessor includes a `sparse` JSON object describing the elements that are different from their initialization values. The `sparse` object contains the following **REQUIRED** properties:

- `count`: number of displaced elements. This number **MUST NOT** be greater than the number of the base accessor elements.
- `indices`: object describing the location and the component type of indices of values to be replaced. The indices **MUST** form a strictly increasing sequence. The indices **MUST NOT** be greater than or equal to the number of the base accessor elements.
- `values`: object describing the location of displaced elements corresponding to the indices referred from the `indices`.

The following example shows an example of a sparse accessor with 10 elements that are different from the initialization array.

```
{
    "accessors": [
        {
            "bufferView": 0,
            "byteOffset": 0,
            "componentType": 5123,
            "count": 12636,
            "type": "VEC3",
            "sparse": {
                "count": 10,
                "indices": {
                    "bufferView": 1,
                    "byteOffset": 0,
                    "componentType": 5123
                },
                "values": {
                    "bufferView": 2,
                    "byteOffset": 0
                }
            }
        }
    ]
}
```


3.6.2.4. Data Alignment

The offset of an `accessor` into a `bufferView` (i.e., `accessor.byteOffset`) and the offset of an `accessor` into a `buffer` (i.e., `accessor.byteOffset + bufferView.byteOffset`) **MUST** be a multiple of the size of the accessor's component type.

When `byteStride` of the referenced `bufferView` is not defined, it means that accessor elements are tightly packed, i.e., effective stride equals the size of the element. When `byteStride` is defined, it **MUST** be a multiple of the size of the accessor's component type.

When two or more vertex attribute accessors use the same `bufferView`, its `byteStride` **MUST** be defined.

Each accessor **MUST** fit its `bufferView`, i.e.,

```
accessor.byteOffset + EFFECTIVE_BYTE_STRIDE * (accessor.count - 1) + SIZE_OF_COMPONENT
* NUMBER_OF_COMPONENTS
```

MUST be less than or equal to `bufferView.length`.

For performance and compatibility reasons, each element of a vertex attribute **MUST** be aligned to 4-byte boundaries inside a `bufferView` (i.e., `accessor.byteOffset` and `bufferView.byteStride` **MUST** be multiples of 4).

Accessors of matrix type have data stored in column-major order; start of each column **MUST** be aligned to 4-byte boundaries. Specifically, when `ROWS * SIZE_OF_COMPONENT` (where `ROWS` is the number of rows of the matrix) is not a multiple of 4, then `(ROWS * SIZE_OF_COMPONENT) % 4` padding bytes **MUST** be inserted at the end of each column.

Only the following three accessor configurations require padding.

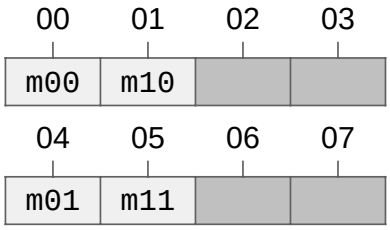

Figure 3. Matrix 2x2, 1-byte Components

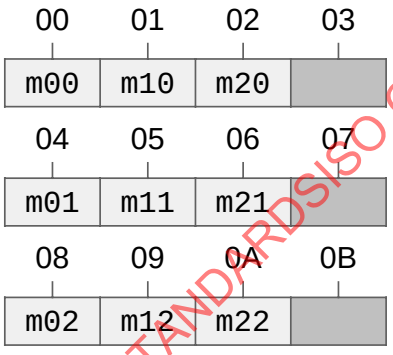

Figure 4. Matrix 3x3, 1-byte Components

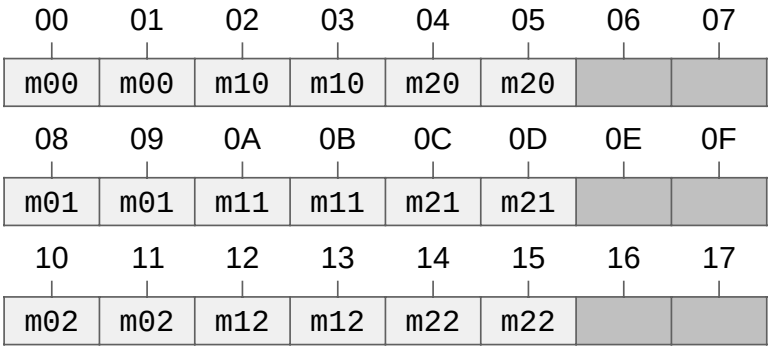

Figure 5. Matrix 3x3, 2-byte Components

Alignment requirements apply only to the start of each column, so trailing bytes **MAY** be omitted if there's no further data.

> *Implementation Note*
>
> Alignment requirements allow client implementations to more efficiently process binary buffers because creating aligned data views usually does not require extra copying.

Consider the following example:

```
{
    "bufferViews": [
        {
            "buffer": 0,
            "byteLength": 17136,
            "byteOffset": 620
        }
    ],
    "accessors": [
        {
            "bufferView": 0,
            "byteOffset": 4608,
            "componentType": 5123,
            "count": 42,
            "type": "VEC2"
        }
    ]
}
```

In this example, the accessor describes tightly-packed two-component unsigned short values.

The corresponding segment of the underlying buffer would start from byte 5228

```
start = accessor.byteOffset + accessor.bufferView.byteOffset
```

and continue until byte 5396 exclusive

```
end = 2 * 2 * accessor.count + start
```

The unsigned short view for the resulting buffer range could be created without copying: 84 scalar values starting from byte offset 5228.

When accessor values are not tightly-packed (i.e., `bufferView.byteStride` is greater than element's byte length), iteration over the created data view would need to take interleaved values into account (i.e., skip them).

3.6.2.5. Accessors Bounds

`accessor.min` and `accessor.max` properties are arrays that contain per-component minimum and maximum values, respectively. The length of these arrays **MUST** be equal to the number of accessor's components.

Values stored in glTF JSON **MUST** match actual minimum and maximum binary values stored in buffers. The `accessor.normalized` flag has no effect on these properties.

A sparse accessor `min` and `max` properties correspond, respectively, to the minimum and maximum component values once the sparse substitution is applied.

When neither `sparse` nor `bufferView` is defined, `min` and `max` properties **MAY** have any values. This is intended for use cases when binary data is supplied by external means (e.g., via extensions).

For floating-point components, JSON-stored minimum and maximum values represent single precision floats and **SHOULD** be rounded to single precision before usage to avoid any potential boundary mismatches.

ECMAScript Implementation Note

`Math.fround` function could be used to achieve that.

Animation input and vertex position attribute accessors **MUST** have `accessor.min` and `accessor.max` defined. For all other accessors, these properties are optional.

3.7. Geometry

3.7.1. Overview

Any node **MAY** contain one mesh, defined in its `mesh` property. The mesh **MAY** be skinned using information provided in a referenced `skin` object. The mesh **MAY** have morph targets.

3.7.2. Meshes

3.7.2.1. Overview

Meshes are defined as arrays of *primitives*. Primitives correspond to the data required for GPU draw calls. Primitives specify one or more `attributes`, corresponding to the vertex attributes used in the draw calls. Indexed primitives also define an `indices` property. Attributes and indices are

defined as references to accessors containing corresponding data. Each primitive **MAY** also specify a `material` and a `mode` that corresponds to the GPU topology type (e.g., triangle set).

Implementation Note

Splitting one mesh into several **primitives** can be useful to limit the number of indices per draw call or to assign different materials to different parts of the mesh.

If `material` is undefined, then a default material **MUST** be used.

The following example defines a mesh containing one indexed triangle primitive:

```
{
    "meshes": [
        {
            "primitives": [
                {
                    "attributes": {
                        "NORMAL": 23,
                        "POSITION": 22,
                        "TANGENT": 24,
                        "TEXCOORD_0": 25
                    },
                    "indices": 21,
                    "material": 3,
                    "mode": 4
                }
            ]
        }
    ]
}
```


Each attribute is defined as a property of the `attributes` object. The name of the property corresponds to an enumerated value identifying the vertex attribute, such as `POSITION`. The value of the property is the index of an accessor that contains the data.

The specification defines the following attribute semantics: `POSITION`, `NORMAL`, `TANGENT`, `TEXCOORD_n`, `COLOR_n`, `JOINTS_n`, and `WEIGHTS_n`.

Application-specific attribute semantics **MUST** start with an underscore, e.g., `_TEMPERATURE`. Application-specific attribute semantics **MUST NOT** use *unsigned int* component type.

Valid accessor type and component type for each attribute semantic property are defined below.

Name	Accessor Type(s)	Component Type(s)	Description
`POSITION`	VEC3	*float*	Unitless XYZ vertex positions
`NORMAL`	VEC3	*float*	Normalized XYZ vertex normals

Name	Accessor Type(s)	Component Type(s)	Description
TANGENT	VEC4	*float*	XYZW vertex tangents where the XYZ portion is normalized, and the W component is a sign value (-1 or +1) indicating handedness of the tangent basis
TEXCOORD_n	VEC2	*float* *unsigned byte* normalized *unsigned short* normalized	ST texture coordinates
COLOR_n	VEC3 VEC4	*float* *unsigned byte* normalized *unsigned short* normalized	RGB or RGBA vertex color linear multiplier
JOINTS_n	VEC4	*unsigned byte unsigned short*	See Skinned Mesh Attributes
WEIGHTS_n	VEC4	*float* *unsigned byte* normalized *unsigned short* normalized	See Skinned Mesh Attributes

POSITION accessor **MUST** have its min and max properties defined.

The W component of each TANGENT accessor element **MUST** be set to 1.0 or -1.0.

When a COLOR_n attribute uses an accessor of "VEC3" type, its alpha component **MUST** be assumed to have a value of 1.0.

All components of each COLOR_0 accessor element **MUST** be clamped to [0.0, 1.0] range.

TEXCOORD_n, COLOR_n, JOINTS_n, and WEIGHTS_n attribute semantic property names **MUST** be of the form [semantic]_[set_index], e.g., TEXCOORD_0, TEXCOORD_1, COLOR_0. All indices for indexed attribute semantics **MUST** start with 0 and be consecutive positive integers: TEXCOORD_0, TEXCOORD_1, etc. Indices **MUST NOT** use leading zeroes to pad the number of digits (e.g., TEXCOORD_01 is not allowed).

Client implementations **SHOULD** support at least two texture coordinate sets, one vertex color, and one joints/weights set.

All attribute accessors for a given primitive **MUST** have the same count. When indices property is not defined, attribute accessors' count indicates the number of vertices to render; when indices property is defined, it indicates the upper (exclusive) bound on the index values in the indices accessor, i.e., all index values **MUST** be less than attribute accessors' count.

indices accessor **MUST NOT** contain the maximum possible value for the component type used (i.e., 255 for unsigned bytes, 65535 for unsigned shorts, 4294967295 for unsigned ints).

Implementation Note

The maximum values trigger primitive restart in some graphics APIs and would require client implementations to rebuild the index buffer.

When `indices` property is not defined, the number of vertex indices to render is defined by `count` of attribute accessors (with the implied values from range `[0..count)`); when `indices` property is defined, the number of vertex indices to render is defined by `count` of accessor referred to by `indices`. In either case, the number of vertex indices **MUST** be valid for the topology type used:

- For *points*, it **MUST** be non-zero.
- For *line loops* and *line strips*, it **MUST** be 2 or greater.
- For *triangle strips* and *triangle fans*, it **MUST** be 3 or greater.
- For *lines*, it **MUST** be divisible by 2 and non-zero.
- For *triangles*, it **MUST** be divisible by 3 and non-zero

Topology types are defined as follows.

- **Points**

  Each vertex defines a single point primitive, according to the equation:

  $$p_i = \{v_i\}$$

- **Line Strips**

  One line primitive is defined by each vertex and the following vertex, according to the equation:

  $$p_i = \{v_i, v_{i+1}\}$$

- **Line Loops**

  Loops are the same as line strips except that a final segment is added from the final specified vertex to the first vertex.

- **Lines**

  Each consecutive pair of vertices defines a single line primitive, according to the equation:

  $$p_i = \{v_{2i}, v_{2i+1}\}$$

- **Triangles**

  Each consecutive set of three vertices defines a single triangle primitive, according to the equation:

  $$p_i = \{v_{3i}, v_{3i+1}, v_{3i+2}\}$$

- **Triangle Strips**

  One triangle primitive is defined by each vertex and the two vertices that follow it, according to

the equation:

$$p_i = \{v_i, v_{i+(1+i\%2)}, v_{i+(2-i\%2)}\}$$

- **Triangle Fans**

  Triangle primitives are defined around a shared common vertex, according to the equation:

$$p_i = \{v_{i+1}, v_{i+2}, v_0\}$$

Mesh geometry **SHOULD NOT** contain degenerate lines or triangles, i.e., lines or triangles that use the same vertex more than once per topology primitive.

When positions are not specified, client implementations **SHOULD** skip primitive's rendering unless its positions are provided by other means (e.g., by an extension). This applies to both indexed and non-indexed geometry.

When tangents are not specified, client implementations **SHOULD** calculate tangents using default MikkTSpace algorithms with the specified vertex positions, normals, and texture coordinates associated with the normal texture.

When normals are not specified, client implementations **MUST** calculate flat normals and the provided tangents (if present) **MUST** be ignored.

Vertices of the same triangle **SHOULD** have the same `tangent.w` value. When vertices of the same triangle have different `tangent.w` values, its tangent space is considered undefined.

The bitangent vectors **MUST** be computed by taking the cross product of the normal and tangent XYZ vectors and multiplying it against the W component of the tangent: `bitangent = cross(normal.xyz, tangent.xyz) * tangent.w`.

Extensions **MAY** add additional attribute names, accessor types, and/or component types.

3.7.2.2. Morph Targets

Morph targets are defined by extending the Mesh concept.

A morph target is a morphable Mesh where the primitives' attributes are obtained by adding the original attributes to a weighted sum of the target's attributes.

For instance, the morph target vertices `POSITION` for the primitive at index **i** are computed in this way:

```
primitives[i].attributes.POSITION +
  weights[0] * primitives[i].targets[0].POSITION +
  weights[1] * primitives[i].targets[1].POSITION +
  weights[2] * primitives[i].targets[2].POSITION + ...
```

Morph targets are specified via the `targets` property defined in the Mesh `primitives`. Each target in the `targets` array is a plain JSON object mapping a primitive attribute to an accessor containing morph target displacement data (deltas).

For each morph target attribute, an original attribute **MUST** be present in the mesh primitive.

Attributes present in the base mesh primitive but not included in a given morph target **MUST** retain their original values for the morph target.

Implementation Note

This allows skipping zero-filled accessors and implies that different morph targets may contain different sets of attributes.

Client implementations **SHOULD** support at least three attributes — `POSITION`, `NORMAL`, and `TANGENT` — for morphing. Client implementations **MAY** optionally support morphed `TEXCOORD_n` and/or `COLOR_n` attributes.

If morph targets contain application-specific semantics, their names **MUST** be prefixed with an underscore (e.g., `_TEMPERATURE`) like the associated attribute semantics.

All primitives **MUST** have the same number of morph targets in the same order.

Accessor type and component type for each morphed attribute semantic property **MUST** follow the table below. Note that the W component for handedness is omitted when targeting `TANGENT` data since handedness cannot be displaced.

Name	Accessor Type(s)	Component Type(s)	Description
`POSITION`	VEC3	*float*	XYZ vertex position displacements
`NORMAL`	VEC3	*float*	XYZ vertex normal displacements
`TANGENT`	VEC3	*float*	XYZ vertex tangent displacements
`TEXCOORD_n`	VEC2	*float* *signed byte* normalized *signed short* normalized *unsigned byte* normalized *unsigned short* normalized	ST texture coordinate displacements

Name	Accessor Type(s)	Component Type(s)	Description
`COLOR_n`	VEC3 VEC4	*float* *signed byte* normalized *signed short* normalized *unsigned byte* normalized *unsigned short* normalized	RGB or RGBA color deltas

`POSITION` accessor **MUST** have its `min` and `max` properties defined.

Displacements for `POSITION`, `NORMAL`, and `TANGENT` attributes **MUST** be applied before any transformation matrices affecting the mesh vertices such as skinning or node transforms.

When the base mesh primitive does not specify tangents, client implementations **SHOULD** calculate tangents for each morph target using default MikkTSpace algorithms with the updated vertex positions, normals, and texture coordinates associated with the normal texture.

When the base mesh primitive does not specify normals, client implementations **MUST** calculate flat normals for each morph target; the provided tangents and their displacements (if present) **MUST** be ignored.

When `COLOR_n` deltas use an accessor of `"VEC3"` type, their alpha components **MUST** be assumed to have a value of `0.0`.

After applying color deltas, all components of each `COLOR_0` morphed accessor element **MUST** be clamped to `[0.0, 1.0]` range.

All morph target accessors **MUST** have the same `count` as the accessors of the original primitive.

A mesh with morph targets **MAY** also define an optional `mesh.weights` property that stores the default targets' weights. These weights **MUST** be used when `node.weights` is undefined. When `mesh.weights` is undefined, the default targets' weights are zeros.

The following example extends the Mesh defined in the previous example to a morphable one by adding two morph targets:

```
{
    "primitives": [
        {
            "attributes": {
                "NORMAL": 23,
                "POSITION": 22,
                "TANGENT": 24,
                "TEXCOORD_0": 25
            },
            "indices": 21,
            "material": 3,
            "targets": [
                {
                    "NORMAL": 33,
                    "POSITION": 32,
                    "TANGENT": 34
                },
                {
                    "NORMAL": 43,
                    "POSITION": 42,
                    "TANGENT": 44
                }
            ]
        }
    ],
    "weights": [0, 0.5]
}
```


The number of morph targets is not limited. Client implementations **SHOULD** support at least eight morphed attributes. This means that they **SHOULD** support eight morph targets when each morph target has one attribute, four morph targets where each morph target has two attributes, or two morph targets where each morph target has three or four attributes.

For assets that contain a higher number of morphed attributes, client implementations **MAY** choose to only use the eight attributes of the morph targets with the highest weights.

Implementation Note

A significant number of authoring and client implementations associate names with morph targets. While the glTF 2.0 specification currently does not provide a way to specify names, most tools use an array of strings, `mesh.extras.targetNames`, for this purpose. The `targetNames` array and all primitive `targets` arrays must have the same length.

3.7.3. Skins

3.7.3.1. Overview

glTF 2.0 meshes support Linear Blend Skinning via skin objects, joint hierarchies, and designated

vertex attributes.

Skins are stored in the `skins` array of the asset. Each skin is defined by a **REQUIRED** `joints` property that lists the indices of nodes used as joints to pose the skin and an **OPTIONAL** `inverseBindMatrices` property that points to an accessor with inverse bind matrices data used to bring coordinates being skinned into the same space as each joint.

The order of joints is defined by the `skin.joints` array and it **MUST** match the order of `inverseBindMatrices` accessor elements (when the latter is present). The `skeleton` property (if present) points to the node that is the common root of a joints hierarchy or to a direct or indirect parent node of the common root.

Implementation Note

Although the `skeleton` property is not needed for computing skinning transforms, it may be used to provide a specific “pivot point” for the skinned geometry.

An accessor referenced by `inverseBindMatrices` **MUST** have floating-point components of `"MAT4"` type. The number of elements of the accessor referenced by `inverseBindMatrices` **MUST** greater than or equal to the number of `joints` elements. The fourth row of each matrix **MUST** be set to `[0.0, 0.0, 0.0, 1.0]`.

Implementation Note

The matrix defining how to pose the skin’s geometry for use with the joints (also known as “Bind Shape Matrix”) should be premultiplied to mesh data or to Inverse Bind Matrices.

3.7.3.2. Joint Hierarchy

The joint hierarchy used for controlling skinned mesh pose is simply the node hierarchy, with each node designated as a *joint* by a reference from the `skin.joints` array. Each skin’s joints **MUST** have a common parent node (direct or indirect) called *common root*, which may or may not be a joint node itself. When a skin is referenced by a node within a scene, the common root **MUST** belong to the same scene.

Implementation Note

A node object does not specify whether it is a joint. Client implementations may need to traverse the `skins` array first, marking each joint node.

A joint node **MAY** have other nodes attached to it, even a complete node sub graph with meshes.

Implementation Note

It’s common to have an entire geometry attached to a joint node without having it being skinned (e.g., a sword attached to a hand). Note that the node transform is the local transform of the node relative to the joint, like any other node in the glTF node hierarchy as described in the Transformations section.

Only the joint transforms are applied to the skinned mesh; the transform of the skinned mesh node

MUST be ignored.

In the example below, the translation of `node_0` and the scale of `node_1` are applied while the translation of `node_3` and rotation of `node_4` are ignored.

```
{
    "nodes": [
        {
            "name": "node_0",
            "children": [ 1 ],
            "translation": [ 0.0, 1.0, 0.0 ]
        },
        {
            "name": "node_1",
            "children": [ 2 ],
            "scale": [ 0.5, 0.5, 0.5 ]
        },
        {
            "name": "node_2"
        },
        {
            "name": "node_3",
            "children": [ 4 ],
            "translation": [ 1.0, 0.0, 0.0 ]
        },
        {
            "name": "node_4",
            "mesh": 0,
            "rotation": [ 0.0, 1.0, 0.0, 0.0 ],
            "skin": 0
        }
    ],
    "skins": [
        {
            "inverseBindMatrices": 0,
            "joints": [ 1, 2 ],
            "skeleton": 1
        }
    ]
}
```


3.7.3.3. Skinned Mesh Attributes

The skinned mesh **MUST** have vertex attributes that are used in skinning calculations. The `JOINTS_n` attribute data contains the indices of the joints from the corresponding `skin.joints` array that affect the vertex. The `WEIGHTS_n` attribute data defines the weights indicating how strongly the joint influences the vertex.

To apply skinning, a transformation matrix is computed for each joint. Then, the per-vertex transformation matrices are computed as weighted linear sums of the joint transformation

matrices. Note that per-joint inverse bind matrices (when present) **MUST** be applied before the base node transforms.

In the following example, a mesh primitive defines `JOINTS_0` and `WEIGHTS_0` vertex attributes:

```
{
    "meshes": [
        {
            "name": "skinned-mesh_1",
            "primitives": [
                {
                    "attributes": {
                        "JOINTS_0": 179,
                        "NORMAL": 165,
                        "POSITION": 163,
                        "TEXCOORD_0": 167,
                        "WEIGHTS_0": 176
                    },
                    "indices": 161,
                    "material": 1,
                    "mode": 4
                }
            ]
        }
    ]
}
```


The number of joints that influence one vertex is limited to 4 per set, so the referenced accessors **MUST** have `VEC4` type and following component types:

- `JOINTS_n`: *unsigned byte* or *unsigned short*
- `WEIGHTS_n`: *float*, or *normalized unsigned byte*, or *normalized unsigned short*

The joint weights for each vertex **MUST NOT** be negative.

Joints **MUST NOT** contain more than one non-zero weight for a given vertex.

When the weights are stored using *float* component type, their linear sum **SHOULD** be as close as reasonably possible to `1.0` for a given vertex.

When the weights are stored using *normalized unsigned byte*, or *normalized unsigned short* component types, their linear sum before normalization **MUST** be `255` or `65535` respectively. Without these requirements, vertices would be deformed significantly because the weight error would get multiplied by the joint position. For example, an error of `1/255` in the weight sum would result in an unacceptably large difference in the joint position.

Implementation Note

The threshold in the official validation tool is set to `2e-7` times the number of non-zero weights per vertex.

Implementation Note

Since the allowed threshold is much lower than minimum possible step for quantized component types, weight sum should be renormalized after quantization.

When any of the vertices are influenced by more than four joints, the additional joint and weight information are stored in subsequent sets. For example, `JOINTS_1` and `WEIGHTS_1` if present will reference the accessor for up to 4 additional joints that influence the vertices. For a given primitive, the number of `JOINTS_n` attribute sets **MUST** be equal to the number of `WEIGHTS_n` attribute sets.

Client implementations **MAY** support only a single set of up to four weights and joints, however not supporting all weight and joint sets present in the file may have an impact on the asset's animation.

All joint values **MUST** be within the range of joints in the skin. Unused joint values (i.e. joints with a weight of zero) **SHOULD** be set to zero.

3.7.4. Instantiation

A mesh is instantiated by `node.mesh` property. The same mesh could be used by many nodes, which could have different transforms. For example:

```
{
    "nodes": [
        {
            "mesh": 11
        },
        {
            "mesh": 11,
            "translation": [
                -20,
                -1,
                0
            ]
        }
    ]
}
```


After applying the node's global transform, mesh vertex position values are meters.

When a mesh primitive uses any triangle-based topology (i.e., *triangles*, *triangle strip*, or *triangle fan*), the determinant of the node's global transform defines the winding order of that primitive. If the determinant is a positive value, the winding order triangle faces is counterclockwise; in the opposite case, the winding order is clockwise.

Implementation Note

Switching the winding order to clockwise enables mirroring geometry via negative scale transforms.

When an instantiated mesh has morph targets, it **MUST** use morph weights specified with the `node.weights` property. When the latter is undefined, `mesh.weights` property **MUST** be used instead. When both of these fields are undefined, the mesh is instantiated in a non-morphed state (i.e., with all morph weights set to zeros).

The example below instantiates a Morph Target with non-default weights.

```
{
    "nodes": [
        {
            "mesh": 11,
            "weights": [0, 0.5]
        }
    ]
}
```

A skin is instantiated within a node using a combination of the node's `mesh` and `skin` properties. The mesh for a skin instance is defined in the `mesh` property. The `skin` property contains the index of the skin to instance.

The following example shows a skinned mesh instance: a skin object, a node with a skinned mesh, and two joint nodes.

```
{
    "skins": [
        {
            "inverseBindMatrices": 29,
            "joints": [1, 2]
        }
    ],
    "nodes": [
        {
            "name":"Skinned mesh node",
            "mesh": 0,
            "skin": 0
        },
        {
            "name":"Skeleton root joint",
            "children": [2],
            "rotation": [
                0,
                0,
                0.7071067811865475,
                0.7071067811865476
            ],
            "translation": [
                4.61599,
                -2.032e-06,
                -5.08e-08
            ]
        },
        {
            "name":"Head",
            "translation": [
                8.76635,
                0,
                0
            ]
        }
    ]
}
```


3.8. Texture Data

3.8.1. Overview

glTF 2.0 separates texture access into three distinct types of objects: Textures, Images, and Samplers.

3.8.2. Textures

Textures are stored in the asset's `textures` array. A texture is defined by an image index, denoted by the `source` property and a sampler index (`sampler`). For example:

```
{
    "textures": [
        {
            "sampler": 0,
            "source": 2
        }
    ]
}
```

glTF 2.0 supports only static 2D textures.

When `texture.source` is undefined, the image **SHOULD** be provided by an extension or application-specific means, otherwise the texture object is undefined.

Implementation Note

Client implementations may render such textures with a predefined placeholder image or being filled with some error color (usually magenta).

When `texture.sampler` is undefined, a sampler with repeat wrapping (in both directions) and auto filtering **MUST** be used.

3.8.3. Images

Images referred to by textures are stored in the `images` array of the asset.

Each image contains one of

- a URI (or IRI) to an external file in one of the supported image formats, or
- a Data URI with embedded data, or
- a reference to a `bufferView`; in that case `mimeType` **MUST** be defined.

The following example shows an image pointing to an external PNG image file and another image referencing a `bufferView` with JPEG data.

```
{
    "images": [
        {
            "uri": "duckCM.png"
        },
        {
            "bufferView": 14,
            "mimeType": "image/jpeg"
        }
    ]
}
```

Client implementations **MAY** need to manually determine the media type of some images. In such a case, the following table **SHOULD** be used to check the values of the first few bytes.

Media Type	Pattern Length	Pattern Bytes
image/png	8	0x89 0x50 0x4E 0x47 0x0D 0x0A 0x1A 0x0A
image/jpeg	3	0xFF 0xD8 0xFF

The image data **MUST** match the `image.mimeType` property when the latter is defined.

The origin of the texture coordinates (0, 0) corresponds to the upper left corner of a texture image. This is illustrated in the following figure, where the respective coordinates are shown for all four corners of a normalized texture space:

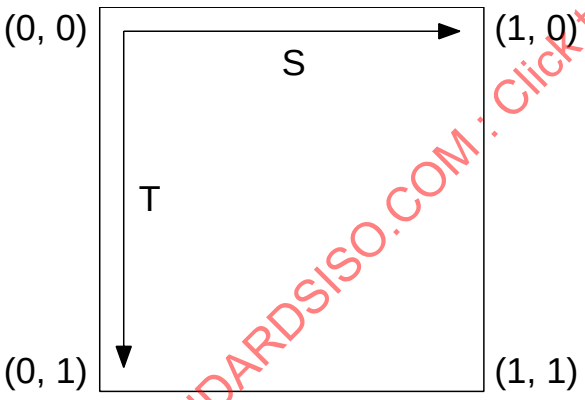

Figure 6. Normalized Texture Coordinates

Any colorspace information (such as ICC profiles, intents, gamma values, etc.) from PNG or JPEG images **MUST** be ignored. Effective transfer function (encoding) is defined by a glTF object that refers to the image (in most cases it's a texture that is used by a material).

Web Implementation Note

To ignore embedded colorspace information when using WebGL API, set `UNPACK_COLORSPACE_CONVERSION_WEBGL` flag to `NONE`.

To ignore embedded colorspace information when using ImageBitmap API, set `colorSpaceConversion` option to `none`.

3.8.4. Samplers

3.8.4.1. Overview

Samplers are stored in the `samplers` array of the asset. Each sampler specifies filtering and wrapping modes.

The sampler properties use integer enums defined in the Properties Reference.

Client implementations **SHOULD** follow specified filtering modes. When the latter are undefined, client implementations **MAY** set their own default texture filtering settings.

Client implementations **MUST** follow specified wrapping modes.

3.8.4.2. Filtering

Filtering modes control texture's magnification and minification.

Magnification modes include:

- *Nearest*. For each requested texel coordinate, the sampler selects a texel with the nearest coordinates. This process is sometimes called "nearest neighbor".
- *Linear*. For each requested texel coordinate, the sampler computes a weighted sum of several adjacent texels. This process is sometimes called "bilinear interpolation".

Minification modes include:

- *Nearest*. For each requested texel coordinate, the sampler selects a texel with the nearest (in Manhattan distance) coordinates from the original image. This process is sometimes called "nearest neighbor".
- *Linear*. For each requested texel coordinate, the sampler computes a weighted sum of several adjacent texels from the original image. This process is sometimes called "bilinear interpolation".
- *Nearest-mipmap-nearest*. For each requested texel coordinate, the sampler first selects one of pre-minified versions of the original image, and then selects a texel with the nearest (in Manhattan distance) coordinates from it.
- *Linear-mipmap-nearest*. For each requested texel coordinate, the sampler first selects one of pre-minified versions of the original image, and then computes a weighted sum of several adjacent texels from it.
- *Nearest-mipmap-linear*. For each requested texel coordinate, the sampler first selects two pre-

minified versions of the original image, selects a texel with the nearest (in Manhattan distance) coordinates from each of them, and performs final linear interpolation between these two intermediate results.

- *Linear-mipmap-linear*. For each requested texel coordinate, the sampler first selects two pre-minified versions of the original image, computes a weighted sum of several adjacent texels from each of them, and performs final linear interpolation between these two intermediate results. This process is sometimes called “trilinear interpolation”.

To properly support mipmap modes, client implementations **SHOULD** generate mipmaps at runtime. When runtime mipmap generation is not possible, client implementations **SHOULD** override the minification filtering mode as follows:

Mipmap minification mode	Fallback mode
Nearest-mipmap-nearest *Nearest-mipmap-linear*	*Nearest*
Linear-mipmap-nearest *Linear-mipmap-linear*	*Linear*

3.8.4.3. Wrapping

Per-vertex texture coordinates, which are provided via `TEXCOORD_n` attribute values, are normalized for the image size (not to confuse with the `normalized` accessor property, the latter refers only to data encoding). That is, the texture coordinate value of `(0.0, 0.0)` points to the beginning of the first (upper-left) image pixel, while the texture coordinate value of `(1.0, 1.0)` points to the end of the last (lower-right) image pixel.

Sampler’s wrapping modes define how to handle texture coordinates that are negative or greater than or equal to `1.0`, independently for both directions. Supported modes include:

- *Repeat*. Only the fractional part of texture coordinates is used.

Example

`2.2` maps to `0.2`; `-0.4` maps to `0.6`.

- *Mirrored Repeat*. This mode works as *repeat* but flips the direction when the integer part (truncated towards −∞) is odd.

Example

`2.2` maps to `0.2`; `-0.4` is treated as `0.4`.

- *Clamp to edge*. Texture coordinates with values outside the image are clamped to the closest existing image texel at the edge.

3.8.4.4. Example

The following example defines a sampler with *linear* magnification filtering, *linear-mipmap-linear* minification filtering, and *repeat* wrapping in both directions.

```
{
    "samplers": [
        {
            "magFilter": 9729,
            "minFilter": 9987,
            "wrapS": 10497,
            "wrapT": 10497
        }
    ]
}
```

3.8.4.5. Non-power-of-two Textures

Client implementations **SHOULD** resize non-power-of-two textures (so that their horizontal and vertical sizes are powers of two) when running on platforms that have limited support for such texture dimensions.

Implementation Note

Specifically, if the `sampler` the texture references:

- has a wrapping mode (either `wrapS` or `wrapT`) equal to *repeat* or *mirrored repeat*, or
- has a minification filter (`minFilter`) that uses mipmapping.

3.9. Materials

3.9.1. Overview

glTF defines materials using a common set of parameters that are based on widely used material representations from Physically Based Rendering (PBR). Specifically, glTF uses the metallic-roughness material model. Using this declarative representation of materials enables a glTF file to be rendered consistently across platforms.

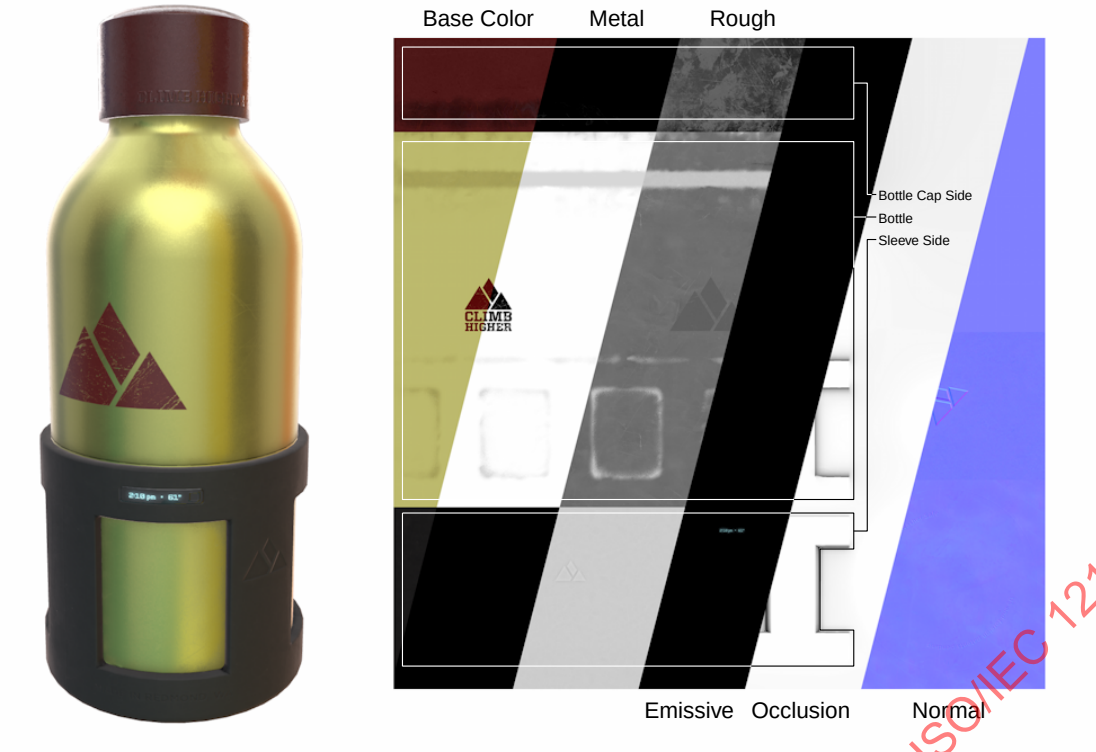

Figure 7. Physically Based Rendering Example

3.9.2. Metallic-Roughness Material

All parameters related to the metallic-roughness material model are defined under the `pbrMetallicRoughness` property of `material` object. The following example shows how to define a gold-like material using the metallic-roughness parameters:

```
{
    "materials": [
        {
            "name": "gold",
            "pbrMetallicRoughness": {
                "baseColorFactor": [ 1.000, 0.766, 0.336, 1.0 ],
                "metallicFactor": 1.0,
                "roughnessFactor": 0.0
            }
        }
    ]
}
```

The metallic-roughness material model is defined by the following properties:

- *base color* - The base color of the material.
- *metalness* - The metalness of the material; values range from `0.0` (non-metal) to `1.0` (metal); see Appendix B for the interpretation of intermediate values.
- *roughness* - The roughness of the material; values range from `0.0` (smooth) to `1.0` (rough).

The *base color* has two different interpretations depending on the value of *metalness*. When the

material is a metal, the base color is the specific measured reflectance value at normal incidence (F0). For a non-metal the base color represents the reflected diffuse color of the material. In this model it is not possible to specify a F0 value for non-metals, and a linear value of 4% (0.04) is used.

The value for each property **MAY** be defined using factors and/or textures (e.g., `baseColorTexture` and `baseColorFactor`). If a texture is not given, all respective texture components within this material model **MUST** be assumed to have a value of `1.0`. If both factors and textures are present, the factor value acts as a linear multiplier for the corresponding texture values. A texture binding is defined by an `index` of a *texture* object and an optional index of texture coordinates.

The following example shows a material that uses a texture for its *base color* property.

```
{
    "materials": [
        {
            "pbrMetallicRoughness": {
                "baseColorTexture": {
                    "index": 0,
                    "texCoord": 1
                },
            }
        }
    ],
    "textures": [
        {
            "source": 0
        }
    ],
    "images": [
        {
            "uri": "base_color.png"
        }
    ]
}
```


The *base color* texture **MUST** contain 8-bit values encoded with the sRGB opto-electronic transfer function so RGB values **MUST** be decoded to real linear values before they are used for any computations. To achieve correct filtering, the transfer function **SHOULD** be decoded before performing linear interpolation.

The textures for *metalness* and *roughness* properties are packed together in a single texture called `metallicRoughnessTexture`. Its *green* channel contains roughness values and its *blue* channel contains metalness values. This texture **MUST** be encoded with linear transfer function and **MAY** use more than 8 bits per channel.

For example, assume an 8-bit RGBA value of `[64, 124, 231, 255]` is sampled from `baseColorTexture` and assume that `baseColorFactor` is given as `[0.2, 1.0, 0.7, 1.0]`. Then, the final *base color* value would be (after decoding the transfer function and multiplying by the factor)

```
[0.051 * 0.2, 0.202 * 1.0, 0.799 * 0.7, 1.0 * 1.0] = [0.0102, 0.202, 0.5593, 1.0]
```

In addition to the material properties, if a primitive specifies a vertex color using the attribute semantic property `COLOR_0`, then this value acts as an additional linear multiplier to *base color*.

Implementations of the bidirectional reflectance distribution function (BRDF) itself **MAY** vary based on device performance and resource constraints. See Appendix B for more details on the BRDF calculations.

3.9.3. Additional Textures

The material definition also provides for additional textures that **MAY** also be used with the metallic-roughness material model as well as other material models, which could be provided via glTF extensions.

The following additional textures are supported:

- **normal** : A tangent space normal texture. The texture encodes XYZ components of a normal vector in tangent space as RGB values stored with linear transfer function. Normal textures **SHOULD NOT** contain *alpha* channel as it not used anyway. After dequantization, texel values **MUST** be mapped as follows: *red* [0.0 .. 1.0] to X [-1 .. 1], *green* [0.0 .. 1.0] to Y [-1 .. 1], *blue* (0.5 .. 1.0] maps to Z (0 .. 1]. Normal textures **SHOULD NOT** contain blue values less than or equal to `0.5`.

  > *Implementation Note*
  >
  > This mapping is usually implemented as `sampledValue * 2.0 - 1.0`.

  The texture binding for normal textures **MAY** additionally contain a scalar `scale` value that linearly scales X and Y components of the normal vector.

  Normal vectors **MUST** be normalized before being used in lighting equations. When scaling is used, vector normalization happens after scaling.

- **occlusion** : The occlusion texture; it indicates areas that receive less indirect lighting from ambient sources. Direct lighting is not affected. The *red* channel of the texture encodes the occlusion value, where `0.0` means fully-occluded area (no indirect lighting) and `1.0` means not occluded area (full indirect lighting). Other texture channels (if present) do not affect occlusion.

  The texture binding for occlusion maps **MAY** optionally contain a scalar `strength` value that is used to reduce the occlusion effect. When present, it affects the occlusion value as `1.0 + strength * (occlusionTexture - 1.0)`.

- **emissive** : The emissive texture and factor control the color and intensity of the light being emitted by the material. The texture **MUST** contain 8-bit values encoded with the sRGB opto-electronic transfer function so RGB values **MUST** be decoded to real linear values before they are used for any computations. To achieve correct filtering, the transfer function **SHOULD** be decoded before performing linear interpolation.

For implementations where a physical light unit is needed, the units for the multiplicative product of the emissive texture and factor are candela per square meter (**cd** / $\mathbf{m}^2$), sometimes called *nits*.

Implementation Note

Because the value is specified per square meter, it indicates the brightness of any given point along the surface. However, the exact conversion from physical light units to the brightness of rendered pixels requires knowledge of the camera's exposure settings, which are left as an implementation detail unless otherwise defined by a glTF extension.

Many rendering engines simplify this calculation by assuming that an emissive factor of `1.0` results in a fully exposed pixel.

The following example shows a material that is defined using `pbrMetallicRoughness` parameters as well as additional textures:

```
{
    "materials": [
        {
            "name": "Material0",
            "pbrMetallicRoughness": {
                "baseColorFactor": [ 0.5, 0.5, 0.5, 1.0 ],
                "baseColorTexture": {
                    "index": 1,
                    "texCoord": 1
                },
                "metallicFactor": 1,
                "roughnessFactor": 1,
                "metallicRoughnessTexture": {
                    "index": 2,
                    "texCoord": 1
                }
            },
            "normalTexture": {
                "scale": 2,
                "index": 3,
                "texCoord": 1
            },
            "emissiveFactor": [ 0.2, 0.1, 0.0 ]
        }
    ]
}
```

If a client implementation is resource-bound and cannot support all the textures defined it **SHOULD** support these additional textures in the following priority order. Resource-bound implementations **SHOULD** drop textures from the bottom to the top.

Texture	Rendering impact when feature is not supported
Normal	Geometry will appear less detailed than authored.
Occlusion	Model will appear brighter in areas that are intended to be darker.
Emissive	Model with lights will not be lit. For example, the headlights of a car model will be off instead of on.

3.9.4. Alpha Coverage

The `alphaMode` property defines how the alpha value is interpreted. The alpha value is taken from the fourth component of the *base color* for metallic-roughness material model.

`alphaMode` can be one of the following values:

- `OPAQUE` - The rendered output is fully opaque and any alpha value is ignored.
- `MASK` - The rendered output is either fully opaque or fully transparent depending on the alpha value and the specified *alpha cutoff* value; the exact appearance of the edges **MAY** be subject to implementation-specific techniques such as “Alpha-to-Coverage”.

Note

This mode is used to simulate geometry such as tree leaves or wire fences.

- `BLEND` - The rendered output is combined with the background using the “over” operator as described in Compositing digital images.

Note

This mode is used to simulate geometry such as gauze cloth or animal fur.

When `alphaMode` is set to `MASK` the `alphaCutoff` property specifies the cutoff threshold. If the alpha value is greater than or equal to the `alphaCutoff` value then it is rendered as fully opaque, otherwise, it is rendered as fully transparent. `alphaCutoff` value is ignored for other modes.

Implementation Note for Real-Time Rasterizers

Real-time rasterizers typically use depth buffers and mesh sorting to support alpha modes. The following describe the expected behavior for these types of renderers.

- `OPAQUE` - A depth value is written for every pixel and mesh sorting is not required for correct output.
- `MASK` - A depth value is not written for a pixel that is discarded after the alpha test. A depth value is written for all other pixels. Mesh sorting is not required for correct output.
- `BLEND` - Support for this mode varies. There is no perfect and fast solution that works for all cases. Client implementations should try to achieve the correct blending output for as many situations as possible. Whether depth value is written or whether to sort is up to the implementation. For example, implementations may discard pixels that have zero or close to zero alpha value to avoid sorting issues.

3.9.5. Double Sided

The `doubleSided` property specifies whether the material is double sided.

When this value is false, back-face culling is enabled, i.e., only front-facing triangles are rendered.

When this value is true, back-face culling is disabled and double sided lighting is enabled. The back-face **MUST** have its normals reversed before the lighting equation is evaluated.

3.9.6. Default Material

The default material, used when a mesh does not specify a material, is defined to be a material with no properties specified. All the default values of `material` apply.

Implementation Note

This material does not emit light and will be black unless some lighting is present in the scene.

3.9.7. Point and Line Materials

This specification does not define size or style of non-triangular primitives (such as *points* or *lines*), and applications **MAY** use various techniques to render these primitives as appropriate. However, the following conventions are **RECOMMENDED** for consistency:

- Points and Lines **SHOULD** have widths of 1px in viewport space.
- Points or Lines with `NORMAL` and `TANGENT` attributes **SHOULD** be rendered with standard lighting including normal textures.
- Points or Lines with `NORMAL` but without `TANGENT` attributes **SHOULD** be rendered with standard lighting but ignoring any normal textures on the material.
- Points or Lines with no `NORMAL` attribute **SHOULD** be rendered without lighting and instead use

the sum of the *base color* value (as defined above, multiplied by `COLOR_0` when present) and the *emissive* value.

3.10. Cameras

3.10.1. Overview

Cameras are stored in the asset's `cameras` array. Each camera defines a `type` property that designates the type of projection (perspective or orthographic), and either a `perspective` or `orthographic` property that defines the details. A camera is instantiated within a node using the `node.camera` property.

A camera object defines the projection matrix that transforms scene coordinates from the view space to the clip space.

A node containing the camera instance defines the view matrix that transforms scene coordinates from the global space to the view space.

3.10.2. View Matrix

The camera is defined such that the local +X axis is to the right, the "lens" looks towards the local -Z axis, and the top of the camera is aligned with the local +Y axis.

The view matrix is derived from the global transform of the node containing the camera with the scaling ignored. If the node's global transform is identity, the location of the camera is at the origin.

3.10.3. Projection Matrices

3.10.3.1. Overview

The projection can be perspective or orthographic.

There are two subtypes of perspective projections: finite and infinite. When the `zfar` property is undefined, the camera defines an infinite projection. Otherwise, the camera defines a finite projection.

The following example defines two perspective cameras with supplied values for Y field of view, aspect ratio, and clipping information.

```
{
    "cameras": [
        {
            "name": "Finite perspective camera",
            "type": "perspective",
            "perspective": {
                "aspectRatio": 1.5,
                "yfov": 0.660593,
                "zfar": 100,
                "znear": 0.01
            }
        },
        {
            "name": "Infinite perspective camera",
            "type": "perspective",
            "perspective": {
                "aspectRatio": 1.5,
                "yfov": 0.660593,
                "znear": 0.01
            }
        }
    ]
}
```

Client implementations **SHOULD** use the following projection matrices.

3.10.3.2. Infinite perspective projection

Let

- `a` be the aspect ratio (width over height) of the field of view, set by `camera.perspective.aspectRatio`, or the aspect ratio of the viewport;
- `y` be the vertical field of view in radians, set by `camera.perspective.yfov`;
- `n` be the distance to the near clipping plane, set by `camera.perspective.znear`.

Then, the projection matrix is defined as follows.

$$\begin{bmatrix} \frac{1}{a \times \tan(0.5 \times y)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(0.5 \times y)} & 0 & 0 \\ 0 & 0 & -1 & -2n \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

When the provided camera's aspect ratio does not match the aspect ratio of the viewport, client implementations **SHOULD NOT** crop or perform non-uniform scaling ("stretching") to fill the viewport.

3.10.3.3. Finite perspective projection

Let

- `a` be the aspect ratio (width over height) of the field of view, set by `camera.perspective.aspectRatio`, or the aspect ratio of the viewport;
- `y` be the vertical field of view in radians, set by `camera.perspective.yfov`;
- `f` be the distance to the far clipping plane, set by `camera.perspective.zfar`;
- `n` be the distance to the near clipping plane, set by `camera.perspective.znear`.

Then, the projection matrix is defined as follows.

$$\begin{bmatrix} \frac{1}{a \times \tan(0.5 \times y)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(0.5 \times y)} & 0 & 0 \\ 0 & 0 & \frac{f+n}{n-f} & \frac{2fn}{n-f} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

When the provided camera's aspect ratio does not match the aspect ratio of the viewport, client implementations **SHOULD NOT** crop or perform non-uniform scaling ("stretching") to fill the viewport.

3.10.3.4. Orthographic projection

Let

- `r` be half the orthographic width, set by `camera.orthographic.xmag`;
- `t` be half the orthographic height, set by `camera.orthographic.ymag`;
- `f` be the distance to the far clipping plane, set by `camera.orthographic.zfar`;
- `n` be the distance to the near clipping plane, set by `camera.orthographic.znear`.

Then, the projection matrix is defined as follows.

$$\begin{bmatrix} \frac{1}{r} & 0 & 0 & 0 \\ 0 & \frac{1}{t} & 0 & 0 \\ 0 & 0 & \frac{2}{n-f} & \frac{f+n}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

When `r / t` does not match the aspect ratio of the viewport, client implementations **SHOULD NOT** crop or perform non-uniform scaling ("stretching") to fill the viewport.

3.11. Animations

glTF supports articulated and skinned animation via key frame animations of nodes' transforms. Key frame data is stored in buffers and referenced in animations using accessors.

glTF 2.0 also supports animation of instantiated morph targets in a similar fashion.

Note

glTF 2.0 only supports animating node transforms and morph target weights. Extensions or a future version of the specification may support animating arbitrary properties, such as material colors and texture transformation matrices.

Note

glTF 2.0 defines only storage of animation keyframes, so this specification doesn't define any runtime behavior, such as: order of playing, auto-start, loops, mapping of timelines, etc. When loading a glTF 2.0 asset, client implementations may select an animation entry and pause it on the first frame, play it automatically, or ignore all animations until further user requests. When a playing animation is stopped, client implementations may reset the scene to the initial state or freeze it at the current frame.

Implementation Note

glTF 2.0 does not specifically define how an animation will be used when imported but, as a best practice, it is recommended that each animation is self-contained as an action. For example, "Walk" and "Run" animations might each contain multiple channels targeting a model's various bones. The client implementation may choose when to play any of the available animations.

All animations are stored in the `animations` array of the asset. An animation is defined as a set of channels (the `channels` property) and a set of samplers that specify accessors with key frame data and interpolation method (the `samplers` property).

The following examples show the expected usage of animations.

```
{
    "animations": [
        {
            "name": "Animate all properties of one node with different samplers",
            "channels": [
                {
                    "sampler": 0,
                    "target": {
                        "node": 1,
                        "path": "rotation"
                    }
                },
                {
                    "sampler": 1,
                    "target": {
                        "node": 1,
                        "path": "scale"
                    }
                },
```

```
            {
                "sampler": 2,
                "target": {
                    "node": 1,
                    "path": "translation"
                }
            }
        ],
        "samplers": [
            {
                "input": 4,
                "interpolation": "LINEAR",
                "output": 5
            },
            {
                "input": 4,
                "interpolation": "LINEAR",
                "output": 6
            },
            {
                "input": 4,
                "interpolation": "LINEAR",
                "output": 7
            }
        ]
    },
    {
        "name": "Animate two nodes with different samplers",
        "channels": [
            {
                "sampler": 0,
                "target": {
                    "node": 0,
                    "path": "rotation"
                }
            },
            {
                "sampler": 1,
                "target": {
                    "node": 1,
                    "path": "rotation"
                }
            }
        ],
        "samplers": [
            {
                "input": 0,
                "interpolation": "LINEAR",
                "output": 1
            },
            {
```

```
                    "input": 2,
                    "interpolation": "LINEAR",
                    "output": 3
                }
            ]
        },
        {
            "name": "Animate two nodes with the same sampler",
            "channels": [
                {
                    "sampler": 0,
                    "target": {
                        "node": 0,
                        "path": "rotation"
                    }
                },
                {
                    "sampler": 0,
                    "target": {
                        "node": 1,
                        "path": "rotation"
                    }
                }
            ],
            "samplers": [
                {
                    "input": 0,
                    "interpolation": "LINEAR",
                    "output": 1
                }
            ]
        },
        {
            "name": "Animate a node rotation channel and the weights of a Morph Target
it instantiates",
            "channels": [
                {
                    "sampler": 0,
                    "target": {
                        "node": 1,
                        "path": "rotation"
                    }
                },
                {
                    "sampler": 1,
                    "target": {
                        "node": 1,
                        "path": "weights"
                    }
                }
            ],
```

```
            "samplers": [
                {
                    "input": 4,
                    "interpolation": "LINEAR",
                    "output": 5
                },
                {
                    "input": 4,
                    "interpolation": "LINEAR",
                    "output": 6
                }
            ]
        }
    ]
}
```

Channels connect the output values of the key frame animation to a specific node in the hierarchy. A channel's `sampler` property contains the index of one of the samplers present in the containing animation's `samplers` array. The `target` property is an object that identifies which node to animate using its `node` property, and which property of the node to animate using `path`. Non-animated properties **MUST** keep their values during animation.

When `node` isn't defined, channel **SHOULD** be ignored. Valid path names are `"translation"`, `"rotation"`, `"scale"`, and `"weights"`.

Nodes that do not contain a mesh with morph targets **MUST NOT** be targeted with `"weights"` path.

Within one animation, each target (a combination of a node and a path) **MUST NOT** be used more than once.

Implementation Note

This prevents potential ambiguities when one target is affected by two or more overlapping samplers.

Each of the animation's **samplers** defines the `input`/`output` pair: a set of floating-point scalar values representing linear time in seconds; and a set of vectors or scalars representing the animated property. All values are stored in a buffer and accessed via accessors; refer to the table below for output accessor types. Interpolation between keys is performed using the interpolation method specified in the `interpolation` property. Supported `interpolation` values include `LINEAR`, `STEP`, and `CUBICSPLINE`. See Appendix C for additional information about interpolation modes.

The inputs of each sampler are relative to `t = 0`, defined as the beginning of the parent `animations` entry. Before and after the provided input range, output **MUST** be clamped to the nearest end of the input range.

Implementation Note

For example, if the earliest sampler input for an animation is `t = 10`, a client implementation must begin playback of that animation channel at `t = 0` with output clamped to the first available output value.

Samplers within a given animation **MAY** have different inputs.

`channel.path`	**Accessor Type**	**Component Type(s)**	**Description**
`"translation"`	`"VEC3"`	*float*	XYZ translation vector
`"rotation"`	`"VEC4"`	*float* *signed byte* normalized *unsigned byte* normalized *signed short* normalized *unsigned short* normalized	XYZW rotation quaternion
`"scale"`	`"VEC3"`	*float*	XYZ scale vector
`"weights"`	`"SCALAR"`	*float* *signed byte* normalized *unsigned byte* normalized *signed short* normalized *unsigned short* normalized	Weights of morph targets

Implementations **MUST** use following equations to decode real floating-point value `f` from a normalized integer `c` and vise-versa:

`accessor.componentType`	**int-to-float**	**float-to-int**
signed byte	`f = max(c / 127.0, -1.0)`	`c = round(f * 127.0)`
unsigned byte	`f = c / 255.0`	`c = round(f * 255.0)`
signed short	`f = max(c / 32767.0, -1.0)`	`c = round(f * 32767.0)`
unsigned short	`f = c / 65535.0`	`c = round(f * 65535.0)`

Animation sampler's `input` accessor **MUST** have its `min` and `max` properties defined.

Implementation Note

Animations with non-linear time inputs, such as time warps in Autodesk 3ds Max or Maya, are not directly representable with glTF animations. glTF is a runtime format and non-linear time inputs are expensive to compute at runtime. Exporter implementations should sample a non-linear time animation into linear inputs and outputs for an accurate representation.

A morph target animation frame is defined by a sequence of scalars of length equal to the number of targets in the animated morph target. These scalar sequences **MUST** lie end-to-end as a single stream in the output accessor, whose final size is equal to the number of morph targets times the number of animation frames.

Morph target animation is by nature sparse, consider using Sparse Accessors for storage of morph target animation. When used with `CUBICSPLINE` interpolation, tangents (a_k, b_k) and values (v_k) are grouped within keyframes:

$a_1,a_2,...a_n,v_1,v_2,...v_n,b_1,b_2,...b_n$

See Appendix C for additional information about interpolation modes.

Skinned animation is achieved by animating the joints in the skin's joint hierarchy.

3.12. Specifying Extensions

glTF defines an extension mechanism that allows the base format to be extended with new capabilities. Any glTF object **MAY** have an optional `extensions` property, as in the following example:

```
{
    "material": [
        {
            "extensions": {
                "KHR_materials_sheen": {
                    "sheenColorFactor": [
                        1.0,
                        0.329,
                        0.1
                    ],
                    "sheenRoughnessFactor": 0.8
                }
            }
        }
    ]
}
```

All extensions used in a glTF asset **MUST** be listed in the top-level `extensionsUsed` array object, e.g.,

```
{
    "extensionsUsed": [
        "KHR_materials_sheen",
        "VENDOR_physics"
    ]
}
```

All glTF extensions required to load and/or render an asset **MUST** be listed in the top-level `extensionsRequired` array, e.g.,

```
{
    "extensionsRequired": [
        "KHR_texture_transform"
    ],
    "extensionsUsed": [
        "KHR_texture_transform"
    ]
}
```

`extensionsRequired` is a subset of `extensionsUsed`. All values in `extensionsRequired` **MUST** also exist in `extensionsUsed`.

Chapter 4. GLB File Format Specification

4.1. General (Informative)

glTF provides two delivery options that can be used together:

- glTF JSON points to external binary data (geometry, key frames, skins), and images.
- glTF JSON embeds base64-encoded binary data, and images inline using data URIs.

Hence, loading glTF files usually requires either separate requests to fetch all binary data, or extra space due to base64-encoding. Base64-encoding requires extra processing to decode and increases the file size (by ~33% for encoded resources). While transport-layer gzip mitigates the file size increase, decompression and decoding still add significant loading time.

To avoid this file size and processing overhead, a container format, *Binary glTF* is introduced that enables a glTF asset, including JSON, buffers, and images, to be stored in a single binary blob.

A Binary glTF asset can still refer to external resources. For example, an application that wants to keep images as separate files may embed everything needed for a scene, except images, in a Binary glTF.

4.2. Structure

A Binary glTF (which can be a file, for example) has the following structure:

- A 12-byte preamble, called the *header*.
- One or more *chunks* that contain JSON content and binary data.

The *chunk* containing JSON **MAY** refer to external resources as usual, and **MAY** also reference resources stored within other *chunks*.

4.3. File Extension & Media Type

The file extension to be used with Binary glTF is `.glb`.

The registered media type is `model/gltf-binary`.

4.4. Binary glTF Layout

4.4.1. Overview

Binary glTF is little endian. The figure below shows an example of a Binary glTF asset.

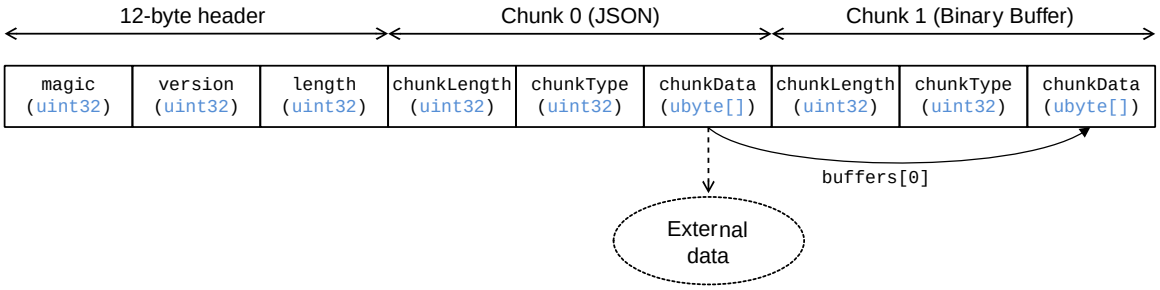

Figure 8. Binary glTF Layout

The following sections describe the structure more in detail.

4.4.2. Header

The 12-byte header consists of three 4-byte entries:

```
uint32 magic
uint32 version
uint32 length
```

- `magic` **MUST** be equal to equal `0x46546C67`. It is ASCII string `glTF` and can be used to identify data as Binary glTF.
- `version` indicates the version of the Binary glTF container format. This specification defines version 2.

  Client implementations that load GLB format **MUST** also check for the asset version properties in the JSON chunk, as the version specified in the GLB header only refers to the GLB container version.

- `length` is the total length of the Binary glTF, including *header* and all *chunks*, in bytes.

4.4.3. Chunks

4.4.3.1. Overview

Each chunk has the following structure:

```
uint32 chunkLength
uint32 chunkType
ubyte[] chunkData
```

- `chunkLength` is the length of `chunkData`, in bytes.
- `chunkType` indicates the type of chunk. See Table 1 for details.
- `chunkData` is the binary payload of the chunk.

The start and the end of each chunk **MUST** be aligned to a 4-byte boundary. See chunks definitions for padding schemes. Chunks **MUST** appear in exactly the order given in Table 1.

Table 1. Chunk types

	Chunk Type	ASCII	Description	Occurrences
1.	0x4E4F534A	JSON	Structured JSON content	1
2.	0x004E4942	BIN	Binary buffer	0 or 1

Client implementations **MUST** ignore chunks with unknown types to enable glTF extensions to reference additional chunks with new types following the first two chunks.

4.4.3.2. Structured JSON Content

This chunk holds the glTF JSON, as it would be provided within a .gltf file.

ECMAScript Implementation Note

In a JavaScript implementation, the `TextDecoder` API can be used to extract the glTF content from the ArrayBuffer, and then the JSON can be parsed with `JSON.parse` as usual.

This chunk **MUST** be the very first chunk of a Binary glTF asset. By reading this chunk first, an implementation is able to progressively retrieve resources from subsequent chunks. This way, it is also possible to read only a selected subset of resources from a Binary glTF asset.

This chunk **MUST** be padded with trailing `Space` chars (`0x20`) to satisfy alignment requirements.

4.4.3.3. Binary buffer

This chunk contains the binary payload for geometry, animation key frames, skins, and images. See GLB-stored Buffer for details on referencing this chunk from JSON.

This chunk **MUST** be the second chunk of the Binary glTF asset.

This chunk **MUST** be padded with trailing zeros (`0x00`) to satisfy alignment requirements.

When the binary buffer is empty or when it is stored by other means, this chunk **SHOULD** be omitted.

Chapter 5. Properties Reference

5.1. Accessor

A typed view into a buffer view that contains raw binary data.

Table 2. `Accessor` *Properties*

	Type	Description	Required
bufferView	`integer`	The index of the bufferView.	No
byteOffset	`integer`	The offset relative to the start of the buffer view in bytes.	No, default: `0`
componentType	`integer`	The datatype of the accessor's components.	✔ Yes
normalized	`boolean`	Specifies whether integer data values are normalized before usage.	No, default: `false`
count	`integer`	The number of elements referenced by this accessor.	✔ Yes
type	`string`	Specifies if the accessor's elements are scalars, vectors, or matrices.	✔ Yes
max	`number [1-16]`	Maximum value of each component in this accessor.	No
min	`number [1-16]`	Minimum value of each component in this accessor.	No
sparse	`accessor.sparse`	Sparse storage of elements that deviate from their initialization value.	No
name	`string`	The user-defined name of this object.	No

	Type	Description	Required
extensions	`extension`	JSON object with extension-specific objects.	No
extras	`extras`	Application-specific data.	No

Additional properties are allowed.

- **JSON schema**: `accessor.schema.json`

5.1.1. accessor.bufferView

The index of the buffer view. When undefined, the accessor **MUST** be initialized with zeros; `sparse` property or extensions **MAY** override zeros with actual values.

- **Type**: `integer`
- **Required**: No
- **Minimum**: `>= 0`

5.1.2. accessor.byteOffset

The offset relative to the start of the buffer view in bytes. This **MUST** be a multiple of the size of the component datatype. This property **MUST NOT** be defined when `bufferView` is undefined.

- **Type**: `integer`
- **Required**: No, default: `0`
- **Minimum**: `>= 0`
- **Related WebGL functions**: `vertexAttribPointer()` offset parameter

5.1.3. accessor.componentType

The datatype of the accessor's components. UNSIGNED_INT type **MUST NOT** be used for any accessor that is not referenced by `mesh.primitive.indices`.

- **Type**: `integer`
- **Required**: ✔ Yes
- **Allowed values**:
  - `5120` BYTE
  - `5121` UNSIGNED_BYTE
  - `5122` SHORT
  - `5123` UNSIGNED_SHORT
  - `5125` UNSIGNED_INT

    - `5126` FLOAT

- **Related WebGL functions**: `type` parameter of `vertexAttribPointer()`. The corresponding typed arrays are `Int8Array`, `Uint8Array`, `Int16Array`, `Uint16Array`, `Uint32Array`, and `Float32Array`.

5.1.4. accessor.normalized

Specifies whether integer data values are normalized (`true`) to [0, 1] (for unsigned types) or to [-1, 1] (for signed types) when they are accessed. This property **MUST NOT** be set to `true` for accessors with `FLOAT` or `UNSIGNED_INT` component type.

- **Type**: `boolean`
- **Required**: No, default: `false`
- **Related WebGL functions**: `normalized` parameter of `vertexAttribPointer()`

5.1.5. accessor.count

The number of elements referenced by this accessor, not to be confused with the number of bytes or number of components.

- **Type**: `integer`
- **Required**: ✔ Yes
- **Minimum**: `>= 1`

5.1.6. accessor.type

Specifies if the accessor's elements are scalars, vectors, or matrices.

- **Type**: `string`
- **Required**: ✔ Yes
- **Allowed values**:
    - `"SCALAR"`
    - `"VEC2"`
    - `"VEC3"`
    - `"VEC4"`
    - `"MAT2"`
    - `"MAT3"`
    - `"MAT4"`

5.1.7. accessor.max

Maximum value of each component in this accessor. Array elements **MUST** be treated as having the same data type as accessor's `componentType`. Both `min` and `max` arrays have the same length. The length is determined by the value of the `type` property; it can be 1, 2, 3, 4, 9, or 16.

`normalized` property has no effect on array values: they always correspond to the actual values stored in the buffer. When the accessor is sparse, this property **MUST** contain maximum values of accessor data with sparse substitution applied.

- **Type**: `number [1-16]`
- **Required**: No

5.1.8. accessor.min

Minimum value of each component in this accessor. Array elements **MUST** be treated as having the same data type as accessor's `componentType`. Both `min` and `max` arrays have the same length. The length is determined by the value of the `type` property; it can be 1, 2, 3, 4, 9, or 16.

`normalized` property has no effect on array values: they always correspond to the actual values stored in the buffer. When the accessor is sparse, this property **MUST** contain minimum values of accessor data with sparse substitution applied.

- **Type**: `number [1-16]`
- **Required**: No

5.1.9. accessor.sparse

Sparse storage of elements that deviate from their initialization value.

- **Type**: `accessor.sparse`
- **Required**: No

5.1.10. accessor.name

The user-defined name of this object. This is not necessarily unique, e.g., an accessor and a buffer could have the same name, or two accessors could even have the same name.

- **Type**: `string`
- **Required**: No

5.1.11. accessor.extensions

JSON object with extension-specific objects.

- **Type**: `extension`
- **Required**: No
- **Type of each property**: Extension

5.1.12. accessor.extras

Application-specific data.

- **Type**: `extras`

- **Required**: No

5.2. Accessor Sparse

Sparse storage of accessor values that deviate from their initialization value.

Table 3. `Accessor Sparse` *Properties*

	Type	Description	Required
count	`integer`	Number of deviating accessor values stored in the sparse array.	✔ Yes
indices	`accessor.sparse.indices`	An object pointing to a buffer view containing the indices of deviating accessor values. The number of indices is equal to `count`. Indices **MUST** strictly increase.	✔ Yes
values	`accessor.sparse.values`	An object pointing to a buffer view containing the deviating accessor values.	✔ Yes
extensions	`extension`	JSON object with extension-specific objects.	No
extras	`extras`	Application-specific data.	No

Additional properties are allowed.

- **JSON schema**: `accessor.sparse.schema.json`

5.2.1. accessor.sparse.count

Number of deviating accessor values stored in the sparse array.

- **Type**: `integer`
- **Required**: ✔ Yes
- **Minimum**: `>= 1`

5.2.2. accessor.sparse.indices

An object pointing to a buffer view containing the indices of deviating accessor values. The number of indices is equal to `count`. Indices **MUST** strictly increase.

- **Type**: `accessor.sparse.indices`
- **Required**: ✔ Yes

5.2.3. accessor.sparse.values

An object pointing to a buffer view containing the deviating accessor values.

- **Type**: `accessor.sparse.values`
- **Required**: ✔ Yes

5.2.4. accessor.sparse.extensions

JSON object with extension-specific objects.

- **Type**: `extension`
- **Required**: No
- **Type of each property**: Extension

5.2.5. accessor.sparse.extras

Application-specific data.

- **Type**: `extras`
- **Required**: No

5.3. Accessor Sparse Indices

An object pointing to a buffer view containing the indices of deviating accessor values. The number of indices is equal to `accessor.sparse.count`. Indices **MUST** strictly increase.

Table 4. `Accessor Sparse Indices` *Properties*

	Type	Description	Required
bufferView	`integer`	The index of the buffer view with sparse indices. The referenced buffer view **MUST NOT** have its `target` or `byteStride` properties defined. The buffer view and the optional `byteOffset` **MUST** be aligned to the `componentType` byte length.	✔ Yes

	Type	Description	Required
byteOffset	`integer`	The offset relative to the start of the buffer view in bytes.	No, default: `0`
componentType	`integer`	The indices data type.	✔ Yes
extensions	`extension`	JSON object with extension-specific objects.	No
extras	`extras`	Application-specific data.	No

Additional properties are allowed.

- **JSON schema**: `accessor.sparse.indices.schema.json`

5.3.1. accessor.sparse.indices.bufferView

The index of the buffer view with sparse indices. The referenced buffer view **MUST NOT** have its `target` or `byteStride` properties defined. The buffer view and the optional `byteOffset` **MUST** be aligned to the `componentType` byte length.

- **Type**: `integer`
- **Required**: ✔ Yes
- **Minimum**: `>= 0`

5.3.2. accessor.sparse.indices.byteOffset

The offset relative to the start of the buffer view in bytes.

- **Type**: `integer`
- **Required**: No, default: `0`
- **Minimum**: `>= 0`

5.3.3. accessor.sparse.indices.componentType

The indices data type.

- **Type**: `integer`
- **Required**: ✔ Yes
- **Allowed values**:
  - `5121` UNSIGNED_BYTE
  - `5123` UNSIGNED_SHORT
  - `5125` UNSIGNED_INT

5.3.4. accessor.sparse.indices.extensions

JSON object with extension-specific objects.

- **Type**: `extension`
- **Required**: No
- **Type of each property**: Extension

5.3.5. accessor.sparse.indices.extras

Application-specific data.

- **Type**: `extras`
- **Required**: No

5.4. Accessor Sparse Values

An object pointing to a buffer view containing the deviating accessor values. The number of elements is equal to `accessor.sparse.count` times number of components. The elements have the same component type as the base accessor. The elements are tightly packed. Data **MUST** be aligned following the same rules as the base accessor.

Table 5. `Accessor Sparse Values` Properties

	Type	Description	Required
bufferView	`integer`	The index of the bufferView with sparse values. The referenced buffer view **MUST NOT** have its `target` or `byteStride` properties defined.	✔ Yes
byteOffset	`integer`	The offset relative to the start of the bufferView in bytes.	No, default: `0`
extensions	`extension`	JSON object with extension-specific objects.	No
extras	`extras`	Application-specific data.	No

Additional properties are allowed.

- **JSON schema**: `accessor.sparse.values.schema.json`

5.4.1. accessor.sparse.values.bufferView

The index of the bufferView with sparse values. The referenced buffer view **MUST NOT** have its `target` or `byteStride` properties defined.

- **Type**: `integer`
- **Required**: ✔ Yes
- **Minimum**: `>= 0`

5.4.2. accessor.sparse.values.byteOffset

The offset relative to the start of the bufferView in bytes.

- **Type**: `integer`
- **Required**: No, default: `0`
- **Minimum**: `>= 0`

5.4.3. accessor.sparse.values.extensions

JSON object with extension-specific objects.

- **Type**: `extension`
- **Required**: No
- **Type of each property**: Extension

5.4.4. accessor.sparse.values.extras

Application-specific data.

- **Type**: `extras`
- **Required**: No

5.5. Animation

A keyframe animation.

Table 6. `Animation` Properties

	Type	Description	Required
channels	`animation.channel [1-*]`	An array of animation channels. An animation channel combines an animation sampler with a target property being animated. Different channels of the same animation **MUST NOT** have the same targets.	✔ Yes
samplers	`animation.sampler [1-*]`	An array of animation samplers. An animation sampler combines timestamps with a sequence of output values and defines an interpolation algorithm.	✔ Yes
name	`string`	The user-defined name of this object.	No
extensions	`extension`	JSON object with extension-specific objects.	No
extras	`extras`	Application-specific data.	No

Additional properties are allowed.

- **JSON schema**: `animation.schema.json`

5.5.1. animation.channels

An array of animation channels. An animation channel combines an animation sampler with a target property being animated. Different channels of the same animation **MUST NOT** have the same targets.

- **Type**: `animation.channel [1-*]`
- **Required**: ✔ Yes

5.5.2. animation.samplers

An array of animation samplers. An animation sampler combines timestamps with a sequence of output values and defines an interpolation algorithm.

- **Type**: `animation.sampler [1-*]`

- **Required**: ✔ Yes

5.5.3. animation.name

The user-defined name of this object. This is not necessarily unique, e.g., an accessor and a buffer could have the same name, or two accessors could even have the same name.

- **Type**: `string`
- **Required**: No

5.5.4. animation.extensions

JSON object with extension-specific objects.

- **Type**: `extension`
- **Required**: No
- **Type of each property**: Extension

5.5.5. animation.extras

Application-specific data.

- **Type**: `extras`
- **Required**: No

5.6. Animation Channel

An animation channel combines an animation sampler with a target property being animated.

Table 7. `Animation Channel` Properties

	Type	Description	Required
sampler	`integer`	The index of a sampler in this animation used to compute the value for the target.	✔ Yes
target	`animation.channel.target`	The descriptor of the animated property.	✔ Yes
extensions	`extension`	JSON object with extension-specific objects.	No
extras	`extras`	Application-specific data.	No

Additional properties are allowed.

- **JSON schema**: `animation.channel.schema.json`

5.6.1. animation.channel.sampler

The index of a sampler in this animation used to compute the value for the target, e.g., a node's translation, rotation, or scale (TRS).

- **Type**: `integer`
- **Required**: ✔ Yes
- **Minimum**: `>= 0`

5.6.2. animation.channel.target

The descriptor of the animated property.

- **Type**: `animation.channel.target`
- **Required**: ✔ Yes

5.6.3. animation.channel.extensions

JSON object with extension-specific objects.

- **Type**: `extension`
- **Required**: No
- **Type of each property**: Extension

5.6.4. animation.channel.extras

Application-specific data.

- **Type**: `extras`
- **Required**: No

5.7. Animation Channel Target

The descriptor of the animated property.

Table 8. `Animation Channel Target` *Properties*

	Type	Description	Required
node	`integer`	The index of the node to animate. When undefined, the animated object **MAY** be defined by an extension.	No
path	`string`	The name of the node's TRS property to animate, or the `"weights"` of the Morph Targets it instantiates. For the `"translation"` property, the values that are provided by the sampler are the translation along the X, Y, and Z axes. For the `"rotation"` property, the values are a quaternion in the order (x, y, z, w), where w is the scalar. For the `"scale"` property, the values are the scaling factors along the X, Y, and Z axes.	✔ Yes
extensions	`extension`	JSON object with extension-specific objects.	No
extras	`extras`	Application-specific data.	No

Additional properties are allowed.

- **JSON schema**: `animation.channel.target.schema.json`

5.7.1. animation.channel.target.node

The index of the node to animate. When undefined, the animated object **MAY** be defined by an extension.

- **Type**: `integer`
- **Required**: No
- **Minimum**: `>= 0`

5.7.2. animation.channel.target.path

The name of the node's TRS property to animate, or the `"weights"` of the Morph Targets it instantiates. For the `"translation"` property, the values that are provided by the sampler are the translation along the X, Y, and Z axes. For the `"rotation"` property, the values are a quaternion in the order (x, y, z, w), where w is the scalar. For the `"scale"` property, the values are the scaling factors along the X, Y, and Z axes.

- **Type**: `string`
- **Required**: ✔ Yes
- **Allowed values**:
  - `"translation"`
  - `"rotation"`
  - `"scale"`
  - `"weights"`

5.7.3. animation.channel.target.extensions

JSON object with extension-specific objects.

- **Type**: `extension`
- **Required**: No
- **Type of each property**: Extension

5.7.4. animation.channel.target.extras

Application-specific data.

- **Type**: `extras`
- **Required**: No

5.8. Animation Sampler

An animation sampler combines timestamps with a sequence of output values and defines an interpolation algorithm.

Table 9. `Animation Sampler` *Properties*

	Type	Description	Required
input	`integer`	The index of an accessor containing keyframe timestamps.	✔ Yes

	Type	Description	Required
interpolation	`string`	Interpolation algorithm.	No, default: `"LINEAR"`
output	`integer`	The index of an accessor, containing keyframe output values.	✔ Yes
extensions	`extension`	JSON object with extension-specific objects.	No
extras	`extras`	Application-specific data.	No

Additional properties are allowed.

- **JSON schema**: `animation.sampler.schema.json`

5.8.1. animation.sampler.input

The index of an accessor containing keyframe timestamps. The accessor **MUST** be of scalar type with floating-point components. The values represent time in seconds with `time[0] >= 0.0`, and strictly increasing values, i.e., `time[n + 1] > time[n]`.

- **Type**: `integer`
- **Required**: ✔ Yes
- **Minimum**: `>= 0`

5.8.2. animation.sampler.interpolation

Interpolation algorithm.

- **Type**: `string`
- **Required**: No, default: `"LINEAR"`
- **Allowed values**:
  - `"LINEAR"` The animated values are linearly interpolated between keyframes. When targeting a rotation, spherical linear interpolation (slerp) **SHOULD** be used to interpolate quaternions. The number of output elements **MUST** equal the number of input elements.
  - `"STEP"` The animated values remain constant to the output of the first keyframe, until the next keyframe. The number of output elements **MUST** equal the number of input elements.
  - `"CUBICSPLINE"` The animation's interpolation is computed using a cubic spline with specified tangents. The number of output elements **MUST** equal three times the number of input elements. For each input element, the output stores three elements, an in-tangent, a spline vertex, and an out-tangent. There **MUST** be at least two keyframes when using this interpolation.

5.8.3. animation.sampler.output

The index of an accessor, containing keyframe output values.

- **Type**: `integer`
- **Required**: ✔ Yes
- **Minimum**: `>= 0`

5.8.4. animation.sampler.extensions

JSON object with extension-specific objects.

- **Type**: `extension`
- **Required**: No
- **Type of each property**: Extension

5.8.5. animation.sampler.extras

Application-specific data.

- **Type**: `extras`
- **Required**: No

5.9. Asset

Metadata about the glTF asset.

Table 10. `Asset` Properties

	Type	Description	Required
copyright	`string`	A copyright message suitable for display to credit the content creator.	No
generator	`string`	Tool that generated this glTF model. Useful for debugging.	No
version	`string`	The glTF version in the form of `<major>.<minor>` that this asset targets.	✔ Yes

	Type	Description	Required
minVersion	`string`	The minimum glTF version in the form of `<major>.<minor>` that this asset targets. This property **MUST NOT** be greater than the asset version.	No
extensions	`extension`	JSON object with extension-specific objects.	No
extras	`extras`	Application-specific data.	No

Additional properties are allowed.

- **JSON schema**: `asset.schema.json`

5.9.1. asset.copyright

A copyright message suitable for display to credit the content creator.

- **Type**: `string`
- **Required**: No

5.9.2. asset.generator

Tool that generated this glTF model. Useful for debugging.

- **Type**: `string`
- **Required**: No

5.9.3. asset.version

The glTF version in the form of `<major>.<minor>` that this asset targets.

- **Type**: `string`
- **Required**: ✔ Yes
- **Pattern**: `^[0-9]+\.[0-9]+$`

5.9.4. asset.minVersion

The minimum glTF version in the form of `<major>.<minor>` that this asset targets. This property **MUST NOT** be greater than the asset version.

- **Type**: `string`

- **Required**: No
- **Pattern**: `^[0-9]+\.[0-9]+$`

5.9.5. asset.extensions

JSON object with extension-specific objects.

- **Type**: `extension`
- **Required**: No
- **Type of each property**: Extension

5.9.6. asset.extras

Application-specific data.

- **Type**: `extras`
- **Required**: No

5.10. Buffer

A buffer points to binary geometry, animation, or skins.

Table 11. `Buffer` Properties

	Type	Description	Required
uri	`string`	The URI (or IRI) of the buffer.	No
byteLength	`integer`	The length of the buffer in bytes.	✔ Yes
name	`string`	The user-defined name of this object.	No
extensions	`extension`	JSON object with extension-specific objects.	No
extras	`extras`	Application-specific data.	No

Additional properties are allowed.

- **JSON schema**: `buffer.schema.json`

5.10.1. buffer.uri

The URI (or IRI) of the buffer. Relative paths are relative to the current glTF asset. Instead of

referencing an external file, this field **MAY** contain a `data:`-URI.

- **Type**: `string`
- **Required**: No
- **Format**: iri-reference

5.10.2. buffer.byteLength

The length of the buffer in bytes.

- **Type**: `integer`
- **Required**: ✔ Yes
- **Minimum**: `>= 1`

5.10.3. buffer.name

The user-defined name of this object. This is not necessarily unique, e.g., an accessor and a buffer could have the same name, or two accessors could even have the same name.

- **Type**: `string`
- **Required**: No

5.10.4. buffer.extensions

JSON object with extension-specific objects.

- **Type**: `extension`
- **Required**: No
- **Type of each property**: Extension

5.10.5. buffer.extras

Application-specific data.

- **Type**: `extras`
- **Required**: No

5.11. Buffer View

A view into a buffer generally representing a subset of the buffer.

Table 12. `Buffer View` Properties

	Type	Description	Required
buffer	`integer`	The index of the buffer.	✔ Yes

	Type	Description	Required
byteOffset	`integer`	The offset into the buffer in bytes.	No, default: `0`
byteLength	`integer`	The length of the bufferView in bytes.	✔ Yes
byteStride	`integer`	The stride, in bytes.	No
target	`integer`	The hint representing the intended GPU buffer type to use with this buffer view.	No
name	`string`	The user-defined name of this object.	No
extensions	`extension`	JSON object with extension-specific objects.	No
extras	`extras`	Application-specific data.	No

Additional properties are allowed.

- **JSON schema**: `bufferView.schema.json`

5.11.1. bufferView.buffer

The index of the buffer.

- **Type**: `integer`
- **Required**: ✔ Yes
- **Minimum**: `>= 0`

5.11.2. bufferView.byteOffset

The offset into the buffer in bytes.

- **Type**: `integer`
- **Required**: No, default: `0`
- **Minimum**: `>= 0`

5.11.3. bufferView.byteLength

The length of the bufferView in bytes.

- **Type**: `integer`
- **Required**: ✔ Yes

- **Minimum**: `>= 1`

5.11.4. bufferView.byteStride

The stride, in bytes, between vertex attributes. When this is not defined, data is tightly packed. When two or more accessors use the same buffer view, this field **MUST** be defined.

- **Type**: `integer`
- **Required**: No
- **Minimum**: `>= 4`
- **Maximum**: `<= 252`
- **Related WebGL functions**: `vertexAttribPointer()` stride parameter

5.11.5. bufferView.target

The hint representing the intended GPU buffer type to use with this buffer view.

- **Type**: `integer`
- **Required**: No
- **Allowed values**:
  - `34962` ARRAY_BUFFER
  - `34963` ELEMENT_ARRAY_BUFFER
- **Related WebGL functions**: `bindBuffer()`

5.11.6. bufferView.name

The user-defined name of this object. This is not necessarily unique, e.g., an accessor and a buffer could have the same name, or two accessors could even have the same name.

- **Type**: `string`
- **Required**: No

5.11.7. bufferView.extensions

JSON object with extension-specific objects.

- **Type**: `extension`
- **Required**: No
- **Type of each property**: Extension

5.11.8. bufferView.extras

Application-specific data.

- **Type**: `extras`

- **Required**: No

5.12. Camera

A camera's projection. A node **MAY** reference a camera to apply a transform to place the camera in the scene.

Table 13. `Camera` Properties

	Type	Description	Required
orthographic	`camera.orthographic`	An orthographic camera containing properties to create an orthographic projection matrix. This property **MUST NOT** be defined when `perspective` is defined.	No
perspective	`camera.perspective`	A perspective camera containing properties to create a perspective projection matrix. This property **MUST NOT** be defined when `orthographic` is defined.	No
type	`string`	Specifies if the camera uses a perspective or orthographic projection.	✔ Yes
name	`string`	The user-defined name of this object.	No
extensions	`extension`	JSON object with extension-specific objects.	No
extras	`extras`	Application-specific data.	No

Additional properties are allowed.

- **JSON schema**: `camera.schema.json`

5.12.1. camera.orthographic

An orthographic camera containing properties to create an orthographic projection matrix. This property **MUST NOT** be defined when `perspective` is defined.

- **Type**: `camera.orthographic`
- **Required**: No

5.12.2. camera.perspective

A perspective camera containing properties to create a perspective projection matrix. This property **MUST NOT** be defined when `orthographic` is defined.

- **Type**: `camera.perspective`
- **Required**: No

5.12.3. camera.type

Specifies if the camera uses a perspective or orthographic projection. Based on this, either the camera's `perspective` or `orthographic` property **MUST** be defined.

- **Type**: `string`
- **Required**: ✔ Yes
- **Allowed values**:
  - `"perspective"`
  - `"orthographic"`

5.12.4. camera.name

The user-defined name of this object. This is not necessarily unique, e.g., an accessor and a buffer could have the same name, or two accessors could even have the same name.

- **Type**: `string`
- **Required**: No

5.12.5. camera.extensions

JSON object with extension-specific objects.

- **Type**: `extension`
- **Required**: No
- **Type of each property**: Extension

5.12.6. camera.extras

Application-specific data.

- **Type**: `extras`
- **Required**: No

5.13. Camera Orthographic

An orthographic camera containing properties to create an orthographic projection matrix.

Table 14. `Camera Orthographic` Properties

	Type	Description	Required
xmag	`number`	The floating-point horizontal magnification of the view. This value **MUST NOT** be equal to zero. This value **SHOULD NOT** be negative.	✔ Yes
ymag	`number`	The floating-point vertical magnification of the view. This value **MUST NOT** be equal to zero. This value **SHOULD NOT** be negative.	✔ Yes
zfar	`number`	The floating-point distance to the far clipping plane. This value **MUST NOT** be equal to zero. `zfar` **MUST** be greater than `znear`.	✔ Yes
znear	`number`	The floating-point distance to the near clipping plane.	✔ Yes
extensions	`extension`	JSON object with extension-specific objects.	No
extras	`extras`	Application-specific data.	No

Additional properties are allowed.

- **JSON schema**: `camera.orthographic.schema.json`

5.13.1. camera.orthographic.xmag

The floating-point horizontal magnification of the view. This value **MUST NOT** be equal to zero. This value **SHOULD NOT** be negative.

- **Type**: `number`

- **Required**: ✔ Yes

5.13.2. camera.orthographic.ymag

The floating-point vertical magnification of the view. This value **MUST NOT** be equal to zero. This value **SHOULD NOT** be negative.

- **Type**: `number`
- **Required**: ✔ Yes

5.13.3. camera.orthographic.zfar

The floating-point distance to the far clipping plane. This value **MUST NOT** be equal to zero. `zfar` **MUST** be greater than `znear`.

- **Type**: `number`
- **Required**: ✔ Yes
- **Minimum**: `> 0`

5.13.4. camera.orthographic.znear

The floating-point distance to the near clipping plane.

- **Type**: `number`
- **Required**: ✔ Yes
- **Minimum**: `>= 0`

5.13.5. camera.orthographic.extensions

JSON object with extension-specific objects.

- **Type**: `extension`
- **Required**: No
- **Type of each property**: Extension

5.13.6. camera.orthographic.extras

Application-specific data.

- **Type**: `extras`
- **Required**: No

5.14. Camera Perspective

A perspective camera containing properties to create a perspective projection matrix.

Table 15. `Camera Perspective` Properties

	Type	Description	Required
aspectRatio	`number`	The floating-point aspect ratio of the field of view.	No
yfov	`number`	The floating-point vertical field of view in radians. This value **SHOULD** be less than π.	✔ Yes
zfar	`number`	The floating-point distance to the far clipping plane.	No
znear	`number`	The floating-point distance to the near clipping plane.	✔ Yes
extensions	`extension`	JSON object with extension-specific objects.	No
extras	`extras`	Application-specific data.	No

Additional properties are allowed.

- **JSON schema**: `camera.perspective.schema.json`

5.14.1. camera.perspective.aspectRatio

The floating-point aspect ratio of the field of view. When undefined, the aspect ratio of the rendering viewport **MUST** be used.

- **Type**: `number`
- **Required**: No
- **Minimum**: `> 0`

5.14.2. camera.perspective.yfov

The floating-point vertical field of view in radians. This value **SHOULD** be less than π.

- **Type**: `number`
- **Required**: ✔ Yes

- **Minimum**: `> 0`

5.14.3. camera.perspective.zfar

The floating-point distance to the far clipping plane. When defined, `zfar` **MUST** be greater than `znear`. If `zfar` is undefined, client implementations **SHOULD** use infinite projection matrix.

- **Type**: `number`
- **Required**: No
- **Minimum**: `> 0`

5.14.4. camera.perspective.znear

The floating-point distance to the near clipping plane.

- **Type**: `number`
- **Required**: ✔ Yes
- **Minimum**: `> 0`

5.14.5. camera.perspective.extensions

JSON object with extension-specific objects.

- **Type**: `extension`
- **Required**: No
- **Type of each property**: Extension

5.14.6. camera.perspective.extras

Application-specific data.

- **Type**: `extras`
- **Required**: No

5.15. Extension

JSON object with extension-specific objects.

Additional properties are allowed.

- **JSON schema**: `extension.schema.json`

5.16. Extras

Application-specific data.

Although `extras` **MAY** have any type, it is common for applications to store and access custom data as key/value pairs. Therefore, `extras` **SHOULD** be a JSON object rather than a primitive value for best portability.

5.17. glTF

The root object for a glTF asset.

Table 16. `glTF` Properties

	Type	Description	Required
extensionsUsed	`string` [1-*]	Names of glTF extensions used in this asset.	No
extensionsRequired	`string` [1-*]	Names of glTF extensions required to properly load this asset.	No
accessors	`accessor` [1-*]	An array of accessors.	No
animations	`animation` [1-*]	An array of keyframe animations.	No
asset	`asset`	Metadata about the glTF asset.	✔ Yes
buffers	`buffer` [1-*]	An array of buffers.	No
bufferViews	`bufferView` [1-*]	An array of bufferViews.	No
cameras	`camera` [1-*]	An array of cameras.	No
images	`image` [1-*]	An array of images.	No
materials	`material` [1-*]	An array of materials.	No
meshes	`mesh` [1-*]	An array of meshes.	No
nodes	`node` [1-*]	An array of nodes.	No
samplers	`sampler` [1-*]	An array of samplers.	No
scene	`integer`	The index of the default scene.	No
scenes	`scene` [1-*]	An array of scenes.	No
skins	`skin` [1-*]	An array of skins.	No
textures	`texture` [1-*]	An array of textures.	No

	Type	Description	Required
extensions	`extension`	JSON object with extension-specific objects.	No
extras	`extras`	Application-specific data.	No

Additional properties are allowed.

- **JSON schema**: `glTF.schema.json`

5.17.1. glTF.extensionsUsed

Names of glTF extensions used in this asset.

- **Type**: `string [1-*]`
  - Each element in the array **MUST** be unique.
- **Required**: No

5.17.2. glTF.extensionsRequired

Names of glTF extensions required to properly load this asset.

- **Type**: `string [1-*]`
  - Each element in the array **MUST** be unique.
- **Required**: No

5.17.3. glTF.accessors

An array of accessors. An accessor is a typed view into a bufferView.

- **Type**: `accessor [1-*]`
- **Required**: No

5.17.4. glTF.animations

An array of keyframe animations.

- **Type**: `animation [1-*]`
- **Required**: No

5.17.5. glTF.asset

Metadata about the glTF asset.

- **Type**: `asset`

- **Required**: ✔ Yes

5.17.6. glTF.buffers

An array of buffers. A buffer points to binary geometry, animation, or skins.

- **Type**: `buffer [1-*]`
- **Required**: No

5.17.7. glTF.bufferViews

An array of bufferViews. A bufferView is a view into a buffer generally representing a subset of the buffer.

- **Type**: `bufferView [1-*]`
- **Required**: No

5.17.8. glTF.cameras

An array of cameras. A camera defines a projection matrix.

- **Type**: `camera [1-*]`
- **Required**: No

5.17.9. glTF.images

An array of images. An image defines data used to create a texture.

- **Type**: `image [1-*]`
- **Required**: No

5.17.10. glTF.materials

An array of materials. A material defines the appearance of a primitive.

- **Type**: `material [1-*]`
- **Required**: No

5.17.11. glTF.meshes

An array of meshes. A mesh is a set of primitives to be rendered.

- **Type**: `mesh [1-*]`
- **Required**: No

5.17.12. glTF.nodes

An array of nodes.

- **Type**: `node [1-*]`
- **Required**: No

5.17.13. glTF.samplers

An array of samplers. A sampler contains properties for texture filtering and wrapping modes.

- **Type**: `sampler [1-*]`
- **Required**: No

5.17.14. glTF.scene

The index of the default scene. This property **MUST NOT** be defined, when `scenes` is undefined.

- **Type**: `integer`
- **Required**: No
- **Minimum**: `>= 0`

5.17.15. glTF.scenes

An array of scenes.

- **Type**: `scene [1-*]`
- **Required**: No

5.17.16. glTF.skins

An array of skins. A skin is defined by joints and matrices.

- **Type**: `skin [1-*]`
- **Required**: No

5.17.17. glTF.textures

An array of textures.

- **Type**: `texture [1-*]`
- **Required**: No

5.17.18. glTF.extensions

JSON object with extension-specific objects.

- **Type**: `extension`

- **Required**: No
- **Type of each property**: Extension

5.17.19. glTF.extras

Application-specific data.

- **Type**: `extras`
- **Required**: No

5.18. Image

Image data used to create a texture. Image **MAY** be referenced by an URI (or IRI) or a buffer view index.

Table 17. `Image` Properties

	Type	Description	Required
uri	`string`	The URI (or IRI) of the image.	No
mimeType	`string`	The image's media type. This field **MUST** be defined when `bufferView` is defined.	No
bufferView	`integer`	The index of the bufferView that contains the image. This field **MUST NOT** be defined when `uri` is defined.	No
name	`string`	The user-defined name of this object.	No
extensions	`extension`	JSON object with extension-specific objects.	No
extras	`extras`	Application-specific data.	No

Additional properties are allowed.

- **JSON schema**: `image.schema.json`

5.18.1. image.uri

The URI (or IRI) of the image. Relative paths are relative to the current glTF asset. Instead of referencing an external file, this field **MAY** contain a `data:`-URI. This field **MUST NOT** be defined when `bufferView` is defined.

- **Type**: `string`
- **Required**: No
- **Format**: iri-reference

5.18.2. image.mimeType

The image's media type. This field **MUST** be defined when `bufferView` is defined.

- **Type**: `string`
- **Required**: No
- **Allowed values**:
  - `"image/jpeg"`
  - `"image/png"`

5.18.3. image.bufferView

The index of the bufferView that contains the image. This field **MUST NOT** be defined when `uri` is defined.

- **Type**: `integer`
- **Required**: No
- **Minimum**: `>= 0`

5.18.4. image.name

The user-defined name of this object. This is not necessarily unique, e.g., an accessor and a buffer could have the same name, or two accessors could even have the same name.

- **Type**: `string`
- **Required**: No

5.18.5. image.extensions

JSON object with extension-specific objects.

- **Type**: `extension`
- **Required**: No
- **Type of each property**: Extension

5.18.6. image.extras

Application-specific data.

- **Type**: `extras`
- **Required**: No

5.19. Material

The material appearance of a primitive.

Table 18. `Material` Properties

	Type	Description	Required
name	`string`	The user-defined name of this object.	No
extensions	`extension`	JSON object with extension-specific objects.	No
extras	`extras`	Application-specific data.	No
pbrMetallicRoughness	`material.pbrMetallicRoughness`	A set of parameter values that are used to define the metallic-roughness material model from Physically Based Rendering (PBR) methodology. When undefined, all the default values of `pbrMetallicRoughness` **MUST** apply.	No
normalTexture	`material.normalTextureInfo`	The tangent space normal texture.	No
occlusionTexture	`material.occlusionTextureInfo`	The occlusion texture.	No
emissiveTexture	`textureInfo`	The emissive texture.	No
emissiveFactor	`number [3]`	The factors for the emissive color of the material.	No, default: `[0,0,0]`
alphaMode	`string`	The alpha rendering mode of the material.	No, default: `"OPAQUE"`

	Type	Description	Required
alphaCutoff	`number`	The alpha cutoff value of the material.	No, default: `0.5`
doubleSided	`boolean`	Specifies whether the material is double sided.	No, default: `false`

Additional properties are allowed.

- **JSON schema**: `material.schema.json`

5.19.1. material.name

The user-defined name of this object. This is not necessarily unique, e.g., an accessor and a buffer could have the same name, or two accessors could even have the same name.

- **Type**: `string`
- **Required**: No

5.19.2. material.extensions

JSON object with extension-specific objects.

- **Type**: `extension`
- **Required**: No
- **Type of each property**: Extension

5.19.3. material.extras

Application-specific data.

- **Type**: `extras`
- **Required**: No

5.19.4. material.pbrMetallicRoughness

A set of parameter values that are used to define the metallic-roughness material model from Physically Based Rendering (PBR) methodology. When undefined, all the default values of `pbrMetallicRoughness` **MUST** apply.

- **Type**: `material.pbrMetallicRoughness`
- **Required**: No

5.19.5. material.normalTexture

The tangent space normal texture. The texture encodes RGB components with linear transfer function. Each texel represents the XYZ components of a normal vector in tangent space. The

normal vectors use the convention +X is right and +Y is up. +Z points toward the viewer. If a fourth component (A) is present, it **MUST** be ignored. When undefined, the material does not have a tangent space normal texture.

- **Type**: `material.normalTextureInfo`
- **Required**: No

5.19.6. material.occlusionTexture

The occlusion texture. The occlusion values are linearly sampled from the R channel. Higher values indicate areas that receive full indirect lighting and lower values indicate no indirect lighting. If other channels are present (GBA), they **MUST** be ignored for occlusion calculations. When undefined, the material does not have an occlusion texture.

- **Type**: `material.occlusionTextureInfo`
- **Required**: No

5.19.7. material.emissiveTexture

The emissive texture. It controls the color and intensity of the light being emitted by the material. This texture contains RGB components encoded with the sRGB transfer function. If a fourth component (A) is present, it **MUST** be ignored. When undefined, the texture **MUST** be sampled as having `1.0` in RGB components.

- **Type**: `textureInfo`
- **Required**: No

5.19.8. material.emissiveFactor

The factors for the emissive color of the material. This value defines linear multipliers for the sampled texels of the emissive texture.

- **Type**: `number [3]`
  - Each element in the array **MUST** be greater than or equal to `0` and less than or equal to `1`.
- **Required**: No, default: `[0,0,0]`

5.19.9. material.alphaMode

The material's alpha rendering mode enumeration specifying the interpretation of the alpha value of the base color.

- **Type**: `string`
- **Required**: No, default: `"OPAQUE"`
- **Allowed values**:
  - `"OPAQUE"` The alpha value is ignored, and the rendered output is fully opaque.
  - `"MASK"` The rendered output is either fully opaque or fully transparent depending on the

alpha value and the specified `alphaCutoff` value; the exact appearance of the edges **MAY** be subject to implementation-specific techniques such as “Alpha-to-Coverage”.

- `"BLEND"` The alpha value is used to composite the source and destination areas. The rendered output is combined with the background using the normal painting operation (i.e. the Porter and Duff over operator).

5.19.10. material.alphaCutoff

Specifies the cutoff threshold when in `MASK` alpha mode. If the alpha value is greater than or equal to this value then it is rendered as fully opaque, otherwise, it is rendered as fully transparent. A value greater than `1.0` will render the entire material as fully transparent. This value **MUST** be ignored for other alpha modes. When `alphaMode` is not defined, this value **MUST NOT** be defined.

- **Type**: `number`
- **Required**: No, default: `0.5`
- **Minimum**: `>= 0`

5.19.11. material.doubleSided

Specifies whether the material is double sided. When this value is false, back-face culling is enabled. When this value is true, back-face culling is disabled and double-sided lighting is enabled. The back-face **MUST** have its normals reversed before the lighting equation is evaluated.

- **Type**: `boolean`
- **Required**: No, default: `false`

5.20. Material Normal Texture Info

Reference to a texture.

Table 19. `Material Normal Texture Info` *Properties*

	Type	Description	Required
index	`integer`	The index of the texture.	✔ Yes
texCoord	`integer`	The set index of texture’s TEXCOORD attribute used for texture coordinate mapping.	No, default: `0`
scale	`number`	The scalar parameter applied to each normal vector of the normal texture.	No, default: `1`

	Type	Description	Required
extensions	`extension`	JSON object with extension-specific objects.	No
extras	`extras`	Application-specific data.	No

Additional properties are allowed.

- **JSON schema**: `material.normalTextureInfo.schema.json`

5.20.1. material.normalTextureInfo.index

The index of the texture.

- **Type**: `integer`
- **Required**: ✔ Yes
- **Minimum**: `>= 0`

5.20.2. material.normalTextureInfo.texCoord

This integer value is used to construct a string in the format `TEXCOORD_<set index>` which is a reference to a key in `mesh.primitives.attributes` (e.g. a value of `0` corresponds to `TEXCOORD_0`). A mesh primitive **MUST** have the corresponding texture coordinate attributes for the material to be applicable to it.

- **Type**: `integer`
- **Required**: No, default: `0`
- **Minimum**: `>= 0`

5.20.3. material.normalTextureInfo.scale

The scalar parameter applied to each normal vector of the texture. This value scales the normal vector in X and Y directions using the formula: `scaledNormal = normalize<sampled normal texture value> * 2.0 - 1.0) * vec3(<normal scale>, <normal scale>, 1.0`.

- **Type**: `number`
- **Required**: No, default: `1`

5.20.4. material.normalTextureInfo.extensions

JSON object with extension-specific objects.

- **Type**: `extension`
- **Required**: No
- **Type of each property**: Extension

5.20.5. material.normalTextureInfo.extras

Application-specific data.

- **Type**: `extras`
- **Required**: No

5.21. Material Occlusion Texture Info

Reference to a texture.

Table 20. `Material Occlusion Texture Info` *Properties*

	Type	Description	Required
index	`integer`	The index of the texture.	✔ Yes
texCoord	`integer`	The set index of texture's TEXCOORD attribute used for texture coordinate mapping.	No, default: `0`
strength	`number`	A scalar multiplier controlling the amount of occlusion applied.	No, default: `1`
extensions	`extension`	JSON object with extension-specific objects.	No
extras	`extras`	Application-specific data.	No

Additional properties are allowed.

- **JSON schema**: `material.occlusionTextureInfo.schema.json`

5.21.1. material.occlusionTextureInfo.index

The index of the texture.

- **Type**: `integer`
- **Required**: ✔ Yes
- **Minimum**: `>= 0`

5.21.2. material.occlusionTextureInfo.texCoord

This integer value is used to construct a string in the format `TEXCOORD_<set index>` which is a

reference to a key in `mesh.primitives.attributes` (e.g. a value of `0` corresponds to `TEXCOORD_0`). A mesh primitive **MUST** have the corresponding texture coordinate attributes for the material to be applicable to it.

- **Type**: `integer`
- **Required**: No, default: `0`
- **Minimum**: `>= 0`

5.21.3. material.occlusionTextureInfo.strength

A scalar parameter controlling the amount of occlusion applied. A value of `0.0` means no occlusion. A value of `1.0` means full occlusion. This value affects the final occlusion value as: `1.0 + strength * (<sampled occlusion texture value> - 1.0)`.

- **Type**: `number`
- **Required**: No, default: `1`
- **Minimum**: `>= 0`
- **Maximum**: `<= 1`

5.21.4. material.occlusionTextureInfo.extensions

JSON object with extension-specific objects.

- **Type**: `extension`
- **Required**: No
- **Type of each property**: Extension

5.21.5. material.occlusionTextureInfo.extras

Application-specific data.

- **Type**: `extras`
- **Required**: No

5.22. Material PBR Metallic Roughness

A set of parameter values that are used to define the metallic-roughness material model from Physically-Based Rendering (PBR) methodology.

Table 21. `Material PBR Metallic Roughness` Properties

	Type	Description	Required
baseColorFactor	`number [4]`	The factors for the base color of the material.	No, default: `[1,1,1,1]`

	Type	Description	Required
baseColorTexture	`textureInfo`	The base color texture.	No
metallicFactor	`number`	The factor for the metalness of the material.	No, default: `1`
roughnessFactor	`number`	The factor for the roughness of the material.	No, default: `1`
metallicRoughnessTexture	`textureInfo`	The metallic-roughness texture.	No
extensions	`extension`	JSON object with extension-specific objects.	No
extras	`extras`	Application-specific data.	No

Additional properties are allowed.

- **JSON schema**: `material.pbrMetallicRoughness.schema.json`

5.22.1. material.pbrMetallicRoughness.baseColorFactor

The factors for the base color of the material. This value defines linear multipliers for the sampled texels of the base color texture.

- **Type**: `number [4]`
  - Each element in the array **MUST** be greater than or equal to `0` and less than or equal to `1`.
- **Required**: No, default: `[1,1,1,1]`

5.22.2. material.pbrMetallicRoughness.baseColorTexture

The base color texture. The first three components (RGB) **MUST** be encoded with the sRGB transfer function. They specify the base color of the material. If the fourth component (A) is present, it represents the linear alpha coverage of the material. Otherwise, the alpha coverage is equal to `1.0`. The `material.alphaMode` property specifies how alpha is interpreted. The stored texels **MUST NOT** be premultiplied. When undefined, the texture **MUST** be sampled as having `1.0` in all components.

- **Type**: `textureInfo`
- **Required**: No

5.22.3. material.pbrMetallicRoughness.metallicFactor

The factor for the metalness of the material. This value defines a linear multiplier for the sampled metalness values of the metallic-roughness texture.

- **Type**: `number`
- **Required**: No, default: `1`
- **Minimum**: `>= 0`
- **Maximum**: `<= 1`

5.22.4. material.pbrMetallicRoughness.roughnessFactor

The factor for the roughness of the material. This value defines a linear multiplier for the sampled roughness values of the metallic-roughness texture.

- **Type**: `number`
- **Required**: No, default: `1`
- **Minimum**: `>= 0`
- **Maximum**: `<= 1`

5.22.5. material.pbrMetallicRoughness.metallicRoughnessTexture

The metallic-roughness texture. The metalness values are sampled from the B channel. The roughness values are sampled from the G channel. These values **MUST** be encoded with a linear transfer function. If other channels are present (R or A), they **MUST** be ignored for metallic-roughness calculations. When undefined, the texture **MUST** be sampled as having `1.0` in G and B components.

- **Type**: `textureInfo`
- **Required**: No

5.22.6. material.pbrMetallicRoughness.extensions

JSON object with extension-specific objects.

- **Type**: `extension`
- **Required**: No
- **Type of each property**: Extension

5.22.7. material.pbrMetallicRoughness.extras

Application-specific data.

- **Type**: `extras`
- **Required**: No

5.23. Mesh

A set of primitives to be rendered. Its global transform is defined by a node that references it.

Table 22. `Mesh` Properties

	Type	Description	Required
primitives	`mesh.primitive [1-*]`	An array of primitives, each defining geometry to be rendered.	✔ Yes
weights	`number [1-*]`	Array of weights to be applied to the morph targets. The number of array elements **MUST** match the number of morph targets.	No
name	`string`	The user-defined name of this object.	No
extensions	`extension`	JSON object with extension-specific objects.	No
extras	`extras`	Application-specific data.	No

Additional properties are allowed.

- **JSON schema**: `mesh.schema.json`

5.23.1. mesh.primitives

An array of primitives, each defining geometry to be rendered.

- **Type**: `mesh.primitive [1-*]`
- **Required**: ✔ Yes

5.23.2. mesh.weights

Array of weights to be applied to the morph targets. The number of array elements **MUST** match the number of morph targets.

- **Type**: `number [1-*]`
- **Required**: No

5.23.3. mesh.name

The user-defined name of this object. This is not necessarily unique, e.g., an accessor and a buffer could have the same name, or two accessors could even have the same name.

- **Type**: `string`
- **Required**: No

5.23.4. mesh.extensions

JSON object with extension-specific objects.

- **Type**: `extension`
- **Required**: No
- **Type of each property**: Extension

5.23.5. mesh.extras

Application-specific data.

- **Type**: `extras`
- **Required**: No

5.24. Mesh Primitive

Geometry to be rendered with the given material.

Related WebGL functions: `drawElements()` and `drawArrays()`

Table 23. `Mesh Primitive` Properties

	Type	Description	Required
attributes	`object`	A plain JSON object, where each key corresponds to a mesh attribute semantic and each value is the index of the accessor containing attribute's data.	✔ Yes
indices	`integer`	The index of the accessor that contains the vertex indices.	No
material	`integer`	The index of the material to apply to this primitive when rendering.	No
mode	`integer`	The topology type of primitives to render.	No, default: `4`

	Type	Description	Required
targets	`object [1-*]`	An array of morph targets.	No
extensions	`extension`	JSON object with extension-specific objects.	No
extras	`extras`	Application-specific data.	No

Additional properties are allowed.

- **JSON schema**: `mesh.primitive.schema.json`

5.24.1. mesh.primitive.attributes

A plain JSON object, where each key corresponds to a mesh attribute semantic and each value is the index of the accessor containing attribute's data.

- **Type**: `object`
- **Required**: ✔ Yes
- **Type of each property**: `integer`

5.24.2. mesh.primitive.indices

The index of the accessor that contains the vertex indices. When this is undefined, the primitive defines non-indexed geometry. When defined, the accessor **MUST** have `SCALAR` type and an unsigned integer component type.

- **Type**: `integer`
- **Required**: No
- **Minimum**: `>= 0`
- **Related WebGL functions**: `drawElements()` when defined and `drawArrays()` otherwise.

5.24.3. mesh.primitive.material

The index of the material to apply to this primitive when rendering.

- **Type**: `integer`
- **Required**: No
- **Minimum**: `>= 0`

5.24.4. mesh.primitive.mode

The topology type of primitives to render.

- **Type**: `integer`
- **Required**: No, default: `4`
- **Allowed values**:
  - `0` POINTS
  - `1` LINES
  - `2` LINE_LOOP
  - `3` LINE_STRIP
  - `4` TRIANGLES
  - `5` TRIANGLE_STRIP
  - `6` TRIANGLE_FAN

5.24.5. mesh.primitive.targets

An array of morph targets.

- **Type**: `object [1-*]`
- **Required**: No

5.24.6. mesh.primitive.extensions

JSON object with extension-specific objects.

- **Type**: `extension`
- **Required**: No
- **Type of each property**: Extension

5.24.7. mesh.primitive.extras

Application-specific data.

- **Type**: `extras`
- **Required**: No

5.25. Node

A node in the node hierarchy. When the node contains `skin`, all `mesh.primitives` **MUST** contain `JOINTS_0` and `WEIGHTS_0` attributes. A node **MAY** have either a `matrix` or any combination of `translation`/`rotation`/`scale` (TRS) properties. TRS properties are converted to matrices and postmultiplied in the `T * R * S` order to compose the transformation matrix; first the scale is applied to the vertices, then the rotation, and then the translation. If none are provided, the transform is the identity. When a node is targeted for animation (referenced by an animation.channel.target), `matrix` **MUST NOT** be present.

Table 24. Node Properties

	Type	Description	Required
camera	`integer`	The index of the camera referenced by this node.	No
children	`integer [1-*]`	The indices of this node's children.	No
skin	`integer`	The index of the skin referenced by this node.	No
matrix	`number [16]`	A floating-point 4x4 transformation matrix stored in column-major order.	No, default: `[1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1]`
mesh	`integer`	The index of the mesh in this node.	No
rotation	`number [4]`	The node's unit quaternion rotation in the order (x, y, z, w), where w is the scalar.	No, default: `[0,0,0,1]`
scale	`number [3]`	The node's non-uniform scale, given as the scaling factors along the x, y, and z axes.	No, default: `[1,1,1]`
translation	`number [3]`	The node's translation along the x, y, and z axes.	No, default: `[0,0,0]`
weights	`number [1-*]`	The weights of the instantiated morph target. The number of array elements **MUST** match the number of morph targets of the referenced mesh. When defined, `mesh` **MUST** also be defined.	No
name	`string`	The user-defined name of this object.	No
extensions	`extension`	JSON object with extension-specific objects.	No

	Type	Description	Required
extras	`extras`	Application-specific data.	No

Additional properties are allowed.

- **JSON schema**: `node.schema.json`

5.25.1. node.camera

The index of the camera referenced by this node.

- **Type**: `integer`
- **Required**: No
- **Minimum**: `>= 0`

5.25.2. node.children

The indices of this node's children.

- **Type**: `integer [1-*]`
  - Each element in the array **MUST** be unique.
  - Each element in the array **MUST** be greater than or equal to `0`.
- **Required**: No

5.25.3. node.skin

The index of the skin referenced by this node. When a skin is referenced by a node within a scene, all joints used by the skin **MUST** belong to the same scene. When defined, `mesh` **MUST** also be defined.

- **Type**: `integer`
- **Required**: No
- **Minimum**: `>= 0`

5.25.4. node.matrix

A floating-point 4x4 transformation matrix stored in column-major order.

- **Type**: `number [16]`
- **Required**: No, default: `[1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1]`
- **Related WebGL functions**: `uniformMatrix4fv()` with the transpose parameter equal to false

5.25.5. node.mesh

The index of the mesh in this node.

- **Type**: `integer`
- **Required**: No
- **Minimum**: `>= 0`

5.25.6. node.rotation

The node's unit quaternion rotation in the order (x, y, z, w), where w is the scalar.

- **Type**: `number [4]`
  - Each element in the array **MUST** be greater than or equal to `-1` and less than or equal to `1`.
- **Required**: No, default: `[0,0,0,1]`

5.25.7. node.scale

The node's non-uniform scale, given as the scaling factors along the x, y, and z axes.

- **Type**: `number [3]`
- **Required**: No, default: `[1,1,1]`

5.25.8. node.translation

The node's translation along the x, y, and z axes.

- **Type**: `number [3]`
- **Required**: No, default: `[0,0,0]`

5.25.9. node.weights

The weights of the instantiated morph target. The number of array elements **MUST** match the number of morph targets of the referenced mesh. When defined, `mesh` **MUST** also be defined.

- **Type**: `number [1-*]`
- **Required**: No

5.25.10. node.name

The user-defined name of this object. This is not necessarily unique, e.g., an accessor and a buffer could have the same name, or two accessors could even have the same name.

- **Type**: `string`
- **Required**: No

5.25.11. node.extensions

JSON object with extension-specific objects.

- **Type**: `extension`
- **Required**: No
- **Type of each property**: Extension

5.25.12. node.extras

Application-specific data.

- **Type**: `extras`
- **Required**: No

5.26. Sampler

Texture sampler properties for filtering and wrapping modes.

Table 25. `Sampler` Properties

	Type	Description	Required
magFilter	`integer`	Magnification filter.	No
minFilter	`integer`	Minification filter.	No
wrapS	`integer`	S (U) wrapping mode.	No, default: `10497`
wrapT	`integer`	T (V) wrapping mode.	No, default: `10497`
name	`string`	The user-defined name of this object.	No
extensions	`extension`	JSON object with extension-specific objects.	No
extras	`extras`	Application-specific data.	No

Additional properties are allowed.

- **JSON schema**: `sampler.schema.json`

5.26.1. sampler.magFilter

Magnification filter.

- **Type**: `integer`
- **Required**: No

- **Allowed values**:
  - `9728` NEAREST
  - `9729` LINEAR
- **Related WebGL functions**: `samplerParameteri()` with pname equal to TEXTURE_MAG_FILTER

5.26.2. sampler.minFilter

Minification filter.

- **Type**: `integer`
- **Required**: No
- **Allowed values**:
  - `9728` NEAREST
  - `9729` LINEAR
  - `9984` NEAREST_MIPMAP_NEAREST
  - `9985` LINEAR_MIPMAP_NEAREST
  - `9986` NEAREST_MIPMAP_LINEAR
  - `9987` LINEAR_MIPMAP_LINEAR
- **Related WebGL functions**: `samplerParameteri()` with pname equal to TEXTURE_MIN_FILTER

5.26.3. sampler.wrapS

S (U) wrapping mode. All valid values correspond to WebGL enums.

- **Type**: `integer`
- **Required**: No, default: `10497`
- **Allowed values**:
  - `33071` CLAMP_TO_EDGE
  - `33648` MIRRORED_REPEAT
  - `10497` REPEAT
- **Related WebGL functions**: `samplerParameteri()` with pname equal to TEXTURE_WRAP_S

5.26.4. sampler.wrapT

T (V) wrapping mode.

- **Type**: `integer`
- **Required**: No, default: `10497`
- **Allowed values**:
  - `33071` CLAMP_TO_EDGE

- `33648` MIRRORED_REPEAT
- `10497` REPEAT

- **Related WebGL functions**: `samplerParameteri()` with pname equal to TEXTURE_WRAP_T

5.26.5. sampler.name

The user-defined name of this object. This is not necessarily unique, e.g., an accessor and a buffer could have the same name, or two accessors could even have the same name.

- **Type**: `string`
- **Required**: No

5.26.6. sampler.extensions

JSON object with extension-specific objects.

- **Type**: `extension`
- **Required**: No
- **Type of each property**: Extension

5.26.7. sampler.extras

Application-specific data.

- **Type**: `extras`
- **Required**: No

5.27. Scene

The root nodes of a scene.

Table 26. `Scene` Properties

	Type	Description	Required
nodes	`integer [1-*]`	The indices of each root node.	No
name	`string`	The user-defined name of this object.	No
extensions	`extension`	JSON object with extension-specific objects.	No
extras	`extras`	Application-specific data.	No

Additional properties are allowed.

- **JSON schema**: `scene.schema.json`

5.27.1. scene.nodes

The indices of each root node.

- **Type**: `integer [1-*]`
  - Each element in the array **MUST** be unique.
  - Each element in the array **MUST** be greater than or equal to `0`.
- **Required**: No

5.27.2. scene.name

The user-defined name of this object. This is not necessarily unique, e.g., an accessor and a buffer could have the same name, or two accessors could even have the same name.

- **Type**: `string`
- **Required**: No

5.27.3. scene.extensions

JSON object with extension-specific objects.

- **Type**: `extension`
- **Required**: No
- **Type of each property**: Extension

5.27.4. scene.extras

Application-specific data.

- **Type**: `extras`
- **Required**: No

5.28. Skin

Joints and matrices defining a skin.

Table 27. `Skin` Properties

	Type	Description	Required
inverseBindMatrices	`integer`	The index of the accessor containing the floating-point 4x4 inverse-bind matrices.	No
skeleton	`integer`	The index of the node used as a skeleton root.	No
joints	`integer [1-*]`	Indices of skeleton nodes, used as joints in this skin.	✔ Yes
name	`string`	The user-defined name of this object.	No
extensions	`extension`	JSON object with extension-specific objects.	No
extras	`extras`	Application-specific data.	No

Additional properties are allowed.

- **JSON schema**: `skin.schema.json`

5.28.1. skin.inverseBindMatrices

The index of the accessor containing the floating-point 4x4 inverse-bind matrices. Its `accessor.count` property **MUST** be greater than or equal to the number of elements of the `joints` array. When undefined, each matrix is a 4x4 identity matrix.

- **Type**: `integer`
- **Required**: No
- **Minimum**: `>= 0`

5.28.2. skin.skeleton

The index of the node used as a skeleton root. The node **MUST** be the closest common root of the joints hierarchy or a direct or indirect parent node of the closest common root.

- **Type**: `integer`
- **Required**: No
- **Minimum**: `>= 0`

5.28.3. skin.joints

Indices of skeleton nodes, used as joints in this skin.

- **Type**: `integer [1-*]`
  - Each element in the array **MUST** be unique.
  - Each element in the array **MUST** be greater than or equal to `0`.
- **Required**: ✔ Yes

5.28.4. skin.name

The user-defined name of this object. This is not necessarily unique, e.g., an accessor and a buffer could have the same name, or two accessors could even have the same name.

- **Type**: `string`
- **Required**: No

5.28.5. skin.extensions

JSON object with extension-specific objects.

- **Type**: `extension`
- **Required**: No
- **Type of each property**: Extension

5.28.6. skin.extras

Application-specific data.

- **Type**: `extras`
- **Required**: No

5.29. Texture

A texture and its sampler.

Related WebGL functions: `createTexture()`, `deleteTexture()`, `bindTexture()`, `texImage2D()`, and `texParameteri()`

Table 28. `Texture` Properties

	Type	Description	Required
sampler	`integer`	The index of the sampler used by this texture. When undefined, a sampler with repeat wrapping and auto filtering **SHOULD** be used.	No

	Type	Description	Required
source	`integer`	The index of the image used by this texture. When undefined, an extension or other mechanism **SHOULD** supply an alternate texture source, otherwise behavior is undefined.	No
name	`string`	The user-defined name of this object.	No
extensions	`extension`	JSON object with extension-specific objects.	No
extras	`extras`	Application-specific data.	No

Additional properties are allowed.

- **JSON schema**: `texture.schema.json`

5.29.1. texture.sampler

The index of the sampler used by this texture. When undefined, a sampler with repeat wrapping and auto filtering **SHOULD** be used.

- **Type**: `integer`
- **Required**: No
- **Minimum**: `>= 0`

5.29.2. texture.source

The index of the image used by this texture. When undefined, an extension or other mechanism **SHOULD** supply an alternate texture source, otherwise behavior is undefined.

- **Type**: `integer`
- **Required**: No
- **Minimum**: `>= 0`

5.29.3. texture.name

The user-defined name of this object. This is not necessarily unique, e.g., an accessor and a buffer could have the same name, or two accessors could even have the same name.

- **Type**: `string`

- **Required**: No

5.29.4. texture.extensions

JSON object with extension-specific objects.

- **Type**: `extension`
- **Required**: No
- **Type of each property**: Extension

5.29.5. texture.extras

Application-specific data.

- **Type**: `extras`
- **Required**: No

5.30. Texture Info

Reference to a texture.

Table 29. `Texture Info` Properties

	Type	Description	Required
index	`integer`	The index of the texture.	✔ Yes
texCoord	`integer`	The set index of texture's TEXCOORD attribute used for texture coordinate mapping.	No, default: `0`
extensions	`extension`	JSON object with extension-specific objects.	No
extras	`extras`	Application-specific data.	No

Additional properties are allowed.

- **JSON schema**: `textureInfo.schema.json`

5.30.1. textureInfo.index

The index of the texture.

- **Type**: `integer`
- **Required**: ✔ Yes
- **Minimum**: `>= 0`

5.30.2. textureInfo.texCoord

This integer value is used to construct a string in the format `TEXCOORD_<set index>` which is a reference to a key in `mesh.primitives.attributes` (e.g. a value of `0` corresponds to `TEXCOORD_0`). A mesh primitive **MUST** have the corresponding texture coordinate attributes for the material to be applicable to it.

- **Type**: `integer`
- **Required**: No, default: `0`
- **Minimum**: `>= 0`

5.30.3. textureInfo.extensions

JSON object with extension-specific objects.

- **Type**: `extension`
- **Required**: No
- **Type of each property**: Extension

5.30.4. textureInfo.extras

Application-specific data.

- **Type**: `extras`
- **Required**: No

Chapter 6. Acknowledgments (Informative)

6.1. Editors

- Saurabh Bhatia, Microsoft
- Patrick Cozzi, Cesium
- Alexey Knyazev, Individual Contributor
- Tony Parisi, Unity

6.2. Khronos 3D Formats Working Group and Alumni

- Remi Arnaud, Vario
- Mike Bond, Adobe
- Leonard Daly, Individual Contributor
- Emiliano Gambaretto, Adobe
- Tobias Häußler, Dassault Systèmes
- Gary Hsu, Microsoft
- Marco Hutter, Individual Contributor
- Uli Klumpp, Individual Contributor
- Max Limper, Fraunhofer IGD
- Ed Mackey, Analytical Graphics, Inc.
- Don McCurdy, Google
- Scott Nagy, Microsoft
- Norbert Nopper, UX3D
- Fabrice Robinet, Individual Contributor (Previous Editor and Incubator)
- Bastian Sdorra, Dassault Systèmes
- Neil Trevett, NVIDIA
- Jan Paul Van Waveren, Oculus
- Amanda Watson, Oculus

6.3. Special Thanks

- Sarah Chow, Cesium
- Tom Fili, Cesium
- Darryl Gough
- Eric Haines, Autodesk
- Yu Chen Hou, Individual Contributor

- Scott Hunter, Analytical Graphics, Inc.
- Brandon Jones, Google
- Arseny Kapoulkine, Individual Contributor
- Jon Leech, Individual Contributor
- Sean Lilley, Cesium
- Juan Linietsky, Godot Engine
- Matthew McMullan, Individual Contributor
- Mohamad Moneimne, University of Pennsylvania
- Kai Ninomiya, formerly Cesium
- Cedric Pinson, Sketchfab
- Jeff Russell, Marmoset
- Miguel Sousa, Fraunhofer IGD
- Timo Sturm, Fraunhofer IGD
- Rob Taglang, Cesium
- Maik Thöner, Fraunhofer IGD
- Steven Vergenz, AltspaceVR
- Corentin Wallez, Google
- Alex Wood, Analytical Graphics, Inc

Appendix A: JSON Schema Reference (Informative)

A.1. JSON Schema for Accessor

```
{
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "$id": "accessor.schema.json",
    "title": "Accessor",
    "type": "object",
    "description": "A typed view into a buffer view that contains raw binary data.",
    "allOf": [ { "$ref": "glTFChildOfRootProperty.schema.json" } ],
    "properties": {
        "bufferView": {
            "allOf": [ { "$ref": "glTFid.schema.json" } ],
            "description": "The index of the bufferView.",
            "gltf_detailedDescription": "The index of the buffer view. When undefined,
the accessor **MUST** be initialized with zeros; `sparse` property or extensions
**MAY** override zeros with actual values."
        },
        "byteOffset": {
            "type": "integer",
            "description": "The offset relative to the start of the buffer view in
bytes.",
            "minimum": 0,
            "default": 0,
            "gltf_detailedDescription": "The offset relative to the start of the
buffer view in bytes.  This **MUST** be a multiple of the size of the component
datatype. This property **MUST NOT** be defined when `bufferView` is undefined.",
            "gltf_webgl": "`vertexAttribPointer()` offset parameter"
        },
        "componentType": {
            "description": "The datatype of the accessor's components.",
            "gltf_detailedDescription": "The datatype of the accessor's components.
UNSIGNED_INT type **MUST NOT** be used for any accessor that is not referenced by
`mesh.primitive.indices`.",
            "gltf_webgl": "`type` parameter of `vertexAttribPointer()`.  The
corresponding typed arrays are `Int8Array`, `Uint8Array`, `Int16Array`, `Uint16Array`,
`Uint32Array`, and `Float32Array`.",
            "anyOf": [
                {
                    "const": 5120,
                    "description": "BYTE",
                    "type": "integer"
                },
                {
                    "const": 5121,
                    "description": "UNSIGNED_BYTE",
```

```
                    "type": "integer"
                },
                {
                    "const": 5122,
                    "description": "SHORT",
                    "type": "integer"
                },
                {
                    "const": 5123,
                    "description": "UNSIGNED_SHORT",
                    "type": "integer"
                },
                {
                    "const": 5125,
                    "description": "UNSIGNED_INT",
                    "type": "integer"
                },
                {
                    "const": 5126,
                    "description": "FLOAT",
                    "type": "integer"
                },
                {
                    "type": "integer"
                }
            ]
        },
        "normalized": {
            "type": "boolean",
            "description": "Specifies whether integer data values are normalized
before usage.",
            "default": false,
            "gltf_detailedDescription": "Specifies whether integer data values are
normalized (`true`) to [0, 1] (for unsigned types) or to [-1, 1] (for signed types)
when they are accessed. This property **MUST NOT** be set to `true` for accessors with
`FLOAT` or `UNSIGNED_INT` component type.",
            "gltf_webgl": "`normalized` parameter of `vertexAttribPointer()` "
        },
        "count": {
            "type": "integer",
            "description": "The number of elements referenced by this accessor.",
            "minimum": 1,
            "gltf_detailedDescription": "The number of elements referenced by this
accessor, not to be confused with the number of bytes or number of components."
        },
        "type": {
            "description": "Specifies if the accessor's elements are scalars, vectors,
or matrices.",
            "anyOf": [
                {
                    "const": "SCALAR"
```

```
                },
                {
                    "const": "VEC2"
                },
                {
                    "const": "VEC3"
                },
                {
                    "const": "VEC4"
                },
                {
                    "const": "MAT2"
                },
                {
                    "const": "MAT3"
                },
                {
                    "const": "MAT4"
                },
                {
                    "type": "string"
                }
            ]
        },
        "max": {
            "type": "array",
            "description": "Maximum value of each component in this accessor.",
            "items": {
                "type": "number"
            },
            "minItems": 1,
            "maxItems": 16,
            "gltf_detailedDescription": "Maximum value of each component in this
accessor.  Array elements **MUST** be treated as having the same data type as
accessor's `componentType`. Both `min` and `max` arrays have the same length.  The
length is determined by the value of the `type` property; it can be 1, 2, 3, 4, 9, or
16.\n\n`normalized` property has no effect on array values: they always correspond to
the actual values stored in the buffer. When the accessor is sparse, this property
**MUST** contain maximum values of accessor data with sparse substitution applied."
        },
        "min": {
            "type": "array",
            "description": "Minimum value of each component in this accessor.",
            "items": {
                "type": "number"
            },
            "minItems": 1,
            "maxItems": 16,
            "gltf_detailedDescription": "Minimum value of each component in this
accessor.  Array elements **MUST** be treated as having the same data type as
accessor's `componentType`. Both `min` and `max` arrays have the same length.  The
```

```
length is determined by the value of the `type` property; it can be 1, 2, 3, 4, 9, or
16.\n\n`normalized` property has no effect on array values: they always correspond to
the actual values stored in the buffer. When the accessor is sparse, this property
**MUST** contain minimum values of accessor data with sparse substitution applied."
        },
        "sparse": {
            "allOf": [ { "$ref": "accessor.sparse.schema.json" } ],
            "description": "Sparse storage of elements that deviate from their
initialization value."
        },
        "name": { },
        "extensions": { },
        "extras": { }
    },
    "dependencies": {
        "byteOffset": [ "bufferView" ]
    },
    "required": [ "componentType", "count", "type" ]
}
```

A.2. JSON Schema for Accessor Sparse

```
{
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "$id": "accessor.sparse.schema.json",
    "title": "Accessor Sparse",
    "type": "object",
    "description": "Sparse storage of accessor values that deviate from their
initialization value.",
    "allOf": [ { "$ref": "glTFProperty.schema.json" } ],
    "properties": {
        "count": {
            "type": "integer",
            "description": "Number of deviating accessor values stored in the sparse
array.",
            "minimum": 1
        },
        "indices": {
            "allOf": [ { "$ref": "accessor.sparse.indices.schema.json" } ],
            "description": "An object pointing to a buffer view containing the indices
of deviating accessor values. The number of indices is equal to `count`. Indices
**MUST** strictly increase."
        },
        "values": {
            "allOf": [ { "$ref": "accessor.sparse.values.schema.json" } ],
            "description": "An object pointing to a buffer view containing the
deviating accessor values."
        },
        "extensions": { },
        "extras": { }
    },
    "required": [ "count", "indices", "values" ]
}
```

A.3. JSON Schema for Accessor Sparse Indices

```
{
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "$id": "accessor.sparse.indices.schema.json",
    "title": "Accessor Sparse Indices",
    "type": "object",
    "description": "An object pointing to a buffer view containing the indices of
deviating accessor values. The number of indices is equal to `accessor.sparse.count`.
Indices **MUST** strictly increase.",
    "allOf": [ { "$ref": "glTFProperty.schema.json" } ],
    "properties": {
        "bufferView": {
            "allOf": [ { "$ref": "glTFid.schema.json" } ],
            "description": "The index of the buffer view with sparse indices. The
referenced buffer view **MUST NOT** have its `target` or `byteStride` properties
defined. The buffer view and the optional `byteOffset` **MUST** be aligned to the
`componentType` byte length."
        },
        "byteOffset": {
            "type": "integer",
            "description": "The offset relative to the start of the buffer view in
bytes.",
            "minimum": 0,
            "default": 0
        },
        "componentType": {
            "description": "The indices data type.",
            "anyOf": [
                {
                    "const": 5121,
                    "description": "UNSIGNED_BYTE",
                    "type": "integer"
                },
                {
                    "const": 5123,
                    "description": "UNSIGNED_SHORT",
                    "type": "integer"
                },
                {
                    "const": 5125,
                    "description": "UNSIGNED_INT",
                    "type": "integer"
                },
                {
                    "type": "integer"
                }
            ]
        },
        "extensions": { },
```

```
            "extras": { }
        },
        "required": [ "bufferView", "componentType" ]
    }
```

A.4. JSON Schema for Accessor Sparse Values

```
{
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "$id": "accessor.sparse.values.schema.json",
    "title": "Accessor Sparse Values",
    "type": "object",
    "description": "An object pointing to a buffer view containing the deviating
accessor values. The number of elements is equal to `accessor.sparse.count` times
number of components. The elements have the same component type as the base accessor.
The elements are tightly packed. Data **MUST** be aligned following the same rules as
the base accessor.",
    "allOf": [ { "$ref": "glTFProperty.schema.json" } ],
    "properties": {
        "bufferView": {
            "allOf": [ { "$ref": "glTFid.schema.json" } ],
            "description": "The index of the bufferView with sparse values. The
referenced buffer view **MUST NOT** have its `target` or `byteStride` properties
defined."
        },
        "byteOffset": {
            "type": "integer",
            "description": "The offset relative to the start of the bufferView in
bytes.",
            "minimum": 0,
            "default": 0
        },
        "extensions": { },
        "extras": { }
    },
    "required": [ "bufferView" ]
}
```

A.5. JSON Schema for Animation

```
{
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "$id": "animation.schema.json",
    "title": "Animation",
    "type": "object",
    "description": "A keyframe animation.",
    "allOf": [ { "$ref": "glTFChildOfRootProperty.schema.json" } ],
    "properties": {
        "channels": {
            "type": "array",
            "description": "An array of animation channels. An animation channel
combines an animation sampler with a target property being animated. Different
channels of the same animation **MUST NOT** have the same targets.",
            "items": {
                "$ref": "animation.channel.schema.json"
            },
            "minItems": 1
        },
        "samplers": {
            "type": "array",
            "description": "An array of animation samplers. An animation sampler
combines timestamps with a sequence of output values and defines an interpolation
algorithm.",
            "items": {
                "$ref": "animation.sampler.schema.json"
            },
            "minItems": 1
        },
        "name": { },
        "extensions": { },
        "extras": { }
    },
    "required": [ "channels", "samplers" ]
}
```

A.6. JSON Schema for Animation Channel

```
{
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "$id": "animation.channel.schema.json",
    "title": "Animation Channel",
    "type": "object",
    "description": "An animation channel combines an animation sampler with a target
property being animated.",
    "allOf": [ { "$ref": "glTFProperty.schema.json" } ],
    "properties": {
        "sampler": {
            "allOf": [ { "$ref": "glTFid.schema.json" } ],
            "description": "The index of a sampler in this animation used to compute
the value for the target.",
            "gltf_detailedDescription": "The index of a sampler in this animation used
to compute the value for the target, e.g., a node's translation, rotation, or scale
(TRS)."
        },
        "target": {
            "allOf": [ { "$ref": "animation.channel.target.schema.json" } ],
            "description": "The descriptor of the animated property."
        },
        "extensions": { },
        "extras": { }
    },
    "required": [ "sampler", "target" ]
}
```

A.7. JSON Schema for Animation Channel Target

```
{
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "$id": "animation.channel.target.schema.json",
    "title": "Animation Channel Target",
    "type": "object",
    "description": "The descriptor of the animated property.",
    "allOf": [ { "$ref": "glTFProperty.schema.json" } ],
    "properties": {
        "node": {
            "allOf": [ { "$ref": "glTFid.schema.json" } ],
            "description": "The index of the node to animate. When undefined, the
animated object **MAY** be defined by an extension."
        },
        "path": {
            "description": "The name of the node's TRS property to animate, or the `
\"weights\"` of the Morph Targets it instantiates. For the `\"translation\"` property,
the values that are provided by the sampler are the translation along the X, Y, and Z
axes. For the `\"rotation\"` property, the values are a quaternion in the order (x, y,
z, w), where w is the scalar. For the `\"scale\"` property, the values are the scaling
factors along the X, Y, and Z axes.",
            "anyOf": [
                {
                    "const": "translation"
                },
                {
                    "const": "rotation"
                },
                {
                    "const": "scale"
                },
                {
                    "const": "weights"
                },
                {
                    "type": "string"
                }
            ]
        },
        "extensions": { },
        "extras": { }
    },
    "required": [ "path" ]
}
```

A.8. JSON Schema for Animation Sampler

```
{
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "$id": "animation.sampler.schema.json",
    "title": "Animation Sampler",
    "type": "object",
    "description": "An animation sampler combines timestamps with a sequence of output
values and defines an interpolation algorithm.",
    "allOf": [ { "$ref": "glTFProperty.schema.json" } ],
    "properties": {
        "input": {
            "allOf": [ { "$ref": "glTFid.schema.json" } ],
            "description": "The index of an accessor containing keyframe timestamps.",
            "gltf_detailedDescription": "The index of an accessor containing keyframe
timestamps. The accessor **MUST** be of scalar type with floating-point components.
The values represent time in seconds with `time[0] >= 0.0`, and strictly increasing
values, i.e., `time[n + 1] > time[n]`."
        },
        "interpolation": {
            "description": "Interpolation algorithm.",
            "default": "LINEAR",
            "gltf_detailedDescription": "Interpolation algorithm.",
            "anyOf": [
                {
                    "const": "LINEAR",
                    "description": "The animated values are linearly interpolated
between keyframes. When targeting a rotation, spherical linear interpolation (slerp)
**SHOULD** be used to interpolate quaternions. The number of output elements **MUST**
equal the number of input elements."
                },
                {
                    "const": "STEP",
                    "description": "The animated values remain constant to the output
of the first keyframe, until the next keyframe. The number of output elements **MUST**
equal the number of input elements."
                },
                {
                    "const": "CUBICSPLINE",
                    "description": "The animation's interpolation is computed using a
cubic spline with specified tangents. The number of output elements **MUST** equal
three times the number of input elements. For each input element, the output stores
three elements, an in-tangent, a spline vertex, and an out-tangent. There **MUST** be
at least two keyframes when using this interpolation."
                },
                {
                    "type": "string"
                }
            ]
        },
```

```
        "output": {
            "allOf": [ { "$ref": "glTFid.schema.json" } ],
            "description": "The index of an accessor, containing keyframe output
values."
        },
        "extensions": { },
        "extras": { }
    },
    "required": [ "input", "output" ]
}
```

A.9. JSON Schema for Asset

```
{
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "$id": "asset.schema.json",
    "title": "Asset",
    "type": "object",
    "description": "Metadata about the glTF asset.",
    "allOf": [ { "$ref": "glTFProperty.schema.json" } ],
    "properties": {
        "copyright": {
            "type": "string",
            "description": "A copyright message suitable for display to credit the
content creator."
        },
        "generator": {
            "type": "string",
            "description": "Tool that generated this glTF model.  Useful for
debugging."
        },
        "version": {
            "type": "string",
            "description": "The glTF version in the form of `<major>.<minor>` that
this asset targets.",
            "pattern": "^[0-9]+\\.[0-9]+$"
        },
        "minVersion": {
            "type": "string",
            "description": "The minimum glTF version in the form of `<major>.<minor>`
that this asset targets. This property **MUST NOT** be greater than the asset
version.",
            "pattern": "^[0-9]+\\.[0-9]+$"
        },
        "extensions": { },
        "extras": { }
    },
    "required": [ "version" ]
}
```

A.10. JSON Schema for Buffer

```
{
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "$id": "buffer.schema.json",
    "title": "Buffer",
    "type": "object",
    "description": "A buffer points to binary geometry, animation, or skins.",
    "allOf": [ { "$ref": "glTFChildOfRootProperty.schema.json" } ],
    "properties": {
        "uri": {
            "type": "string",
            "description": "The URI (or IRI) of the buffer.",
            "format": "iri-reference",
            "gltf_detailedDescription": "The URI (or IRI) of the buffer.  Relative
paths are relative to the current glTF asset.  Instead of referencing an external
file, this field **MAY** contain a `data:`-URI.",
            "gltf_uriType": "application"
        },
        "byteLength": {
            "type": "integer",
            "description": "The length of the buffer in bytes.",
            "minimum": 1
        },
        "name": { },
        "extensions": { },
        "extras": { }
    },
    "required": [ "byteLength" ]
}
```

A.11. JSON Schema for Buffer View

```
{
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "$id": "bufferView.schema.json",
    "title": "Buffer View",
    "type": "object",
    "description": "A view into a buffer generally representing a subset of the
buffer.",
    "allOf": [ { "$ref": "glTFChildOfRootProperty.schema.json" } ],
    "properties": {
        "buffer": {
            "allOf": [ { "$ref": "glTFid.schema.json" } ],
            "description": "The index of the buffer."
        },
        "byteOffset": {
            "type": "integer",
            "description": "The offset into the buffer in bytes.",
            "minimum": 0,
            "default": 0
        },
        "byteLength": {
            "type": "integer",
            "description": "The length of the bufferView in bytes.",
            "minimum": 1
        },
        "byteStride": {
            "type": "integer",
            "description": "The stride, in bytes.",
            "minimum": 4,
            "maximum": 252,
            "multipleOf": 4,
            "gltf_detailedDescription": "The stride, in bytes, between vertex
attributes.  When this is not defined, data is tightly packed. When two or more
accessors use the same buffer view, this field **MUST** be defined.",
            "gltf_webgl": "`vertexAttribPointer()` stride parameter"
        },
        "target": {
            "description": "The hint representing the intended GPU buffer type to use
with this buffer view.",
            "gltf_webgl": "`bindBuffer()`",
            "anyOf": [
                {
                    "const": 34962,
                    "description": "ARRAY_BUFFER",
                    "type": "integer"
                },
                {
                    "const": 34963,
                    "description": "ELEMENT_ARRAY_BUFFER",
```

```
                    "type": "integer"
                },
                {
                    "type": "integer"
                }
            ]
        },
        "name": { },
        "extensions": { },
        "extras": { }
    },
    "required": [ "buffer", "byteLength" ]
}
```

A.12. JSON Schema for Camera

```
{
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "$id": "camera.schema.json",
    "title": "Camera",
    "type": "object",
    "description": "A camera's projection.  A node **MAY** reference a camera to apply
a transform to place the camera in the scene.",
    "allOf": [ { "$ref": "glTFChildOfRootProperty.schema.json" } ],
    "properties": {
        "orthographic": {
            "allOf": [ { "$ref": "camera.orthographic.schema.json" } ],
            "description": "An orthographic camera containing properties to create an
orthographic projection matrix. This property **MUST NOT** be defined when
`perspective` is defined."
        },
        "perspective": {
            "allOf": [ { "$ref": "camera.perspective.schema.json" } ],
            "description": "A perspective camera containing properties to create a
perspective projection matrix. This property **MUST NOT** be defined when
`orthographic` is defined."
        },
        "type": {
            "description": "Specifies if the camera uses a perspective or orthographic
projection.",
            "gltf_detailedDescription": "Specifies if the camera uses a perspective or
orthographic projection.  Based on this, either the camera's `perspective` or
`orthographic` property **MUST** be defined.",
            "anyOf": [
                {
                    "const": "perspective"
                },
                {
                    "const": "orthographic"
                },
                {
                    "type": "string"
                }
            ]
        },
        "name": { },
        "extensions": { },
        "extras": { }
    },
    "required": [ "type" ],
    "not": {
        "required": [ "perspective", "orthographic" ]
    }
}
```

A.13. JSON Schema for Camera Orthographic

```
{
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "$id": "camera.orthographic.schema.json",
    "title": "Camera Orthographic",
    "type": "object",
    "description": "An orthographic camera containing properties to create an
orthographic projection matrix.",
    "allOf": [ { "$ref": "glTFProperty.schema.json" } ],
    "properties": {
        "xmag": {
            "type": "number",
            "description": "The floating-point horizontal magnification of the view.
This value **MUST NOT** be equal to zero. This value **SHOULD NOT** be negative."
        },
        "ymag": {
            "type": "number",
            "description": "The floating-point vertical magnification of the view.
This value **MUST NOT** be equal to zero. This value **SHOULD NOT** be negative."
        },
        "zfar": {
            "type": "number",
            "description": "The floating-point distance to the far clipping plane.
This value **MUST NOT** be equal to zero. `zfar` **MUST** be greater than `znear`.",
            "exclusiveMinimum": 0.0
        },
        "znear": {
            "type": "number",
            "description": "The floating-point distance to the near clipping plane.",
            "minimum": 0.0
        },
        "extensions": { },
        "extras": { }
    },
    "required": [ "xmag", "ymag", "zfar", "znear" ]
}
```

A.14. JSON Schema for Camera Perspective

```
{
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "$id": "camera.perspective.schema.json",
    "title": "Camera Perspective",
    "type": "object",
    "description": "A perspective camera containing properties to create a perspective
projection matrix.",
    "allOf": [ { "$ref": "glTFProperty.schema.json" } ],
    "properties": {
        "aspectRatio": {
            "type": "number",
            "description": "The floating-point aspect ratio of the field of view.",
            "exclusiveMinimum": 0.0,
            "gltf_detailedDescription": "The floating-point aspect ratio of the field
of view. When undefined, the aspect ratio of the rendering viewport **MUST** be used."
        },
        "yfov": {
            "type": "number",
            "description": "The floating-point vertical field of view in radians. This
value **SHOULD** be less than π.",
            "exclusiveMinimum": 0.0
        },
        "zfar": {
            "type": "number",
            "description": "The floating-point distance to the far clipping plane.",
            "exclusiveMinimum": 0.0,
            "gltf_detailedDescription": "The floating-point distance to the far
clipping plane. When defined, `zfar` **MUST** be greater than `znear`. If `zfar` is
undefined, client implementations **SHOULD** use infinite projection matrix."
        },
        "znear": {
            "type": "number",
            "description": "The floating-point distance to the near clipping plane.",
            "exclusiveMinimum": 0.0,
            "gltf_detailedDescription": "The floating-point distance to the near
clipping plane."
        },
        "extensions": { },
        "extras": { }
    },
    "required": [ "yfov", "znear" ]
}
```

A.15. JSON Schema for Extension

```
{
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "$id": "extension.schema.json",
    "title": "Extension",
    "type": "object",
    "description": "JSON object with extension-specific objects.",
    "properties": {
    },
    "additionalProperties": {
        "type": "object"
    }
}
```

A.16. JSON Schema for Extras

```
{
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "$id": "extras.schema.json",
    "title": "Extras",
    "description": "Application-specific data.",
    "gltf_sectionDescription": "Although `extras` **MAY** have any type, it is common
for applications to store and access custom data as key/value pairs. Therefore,
`extras` **SHOULD** be a JSON object rather than a primitive value for best
portability."
}
```

A.17. JSON Schema for glTF

```
{
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "$id": "glTF.schema.json",
    "title": "glTF",
    "type": "object",
    "description": "The root object for a glTF asset.",
    "allOf": [ { "$ref": "glTFProperty.schema.json" } ],
    "properties": {
        "extensionsUsed": {
            "type": "array",
            "description": "Names of glTF extensions used in this asset.",
            "items": {
                "type": "string"
            },
            "uniqueItems": true,
            "minItems": 1
        },
        "extensionsRequired": {
            "type": "array",
            "description": "Names of glTF extensions required to properly load this
asset.",
            "items": {
                "type": "string"
            },
            "uniqueItems": true,
            "minItems": 1
        },
        "accessors": {
            "type": "array",
            "description": "An array of accessors.",
            "items": {
                "$ref": "accessor.schema.json"
            },
            "minItems": 1,
            "gltf_detailedDescription": "An array of accessors.  An accessor is a
typed view into a bufferView."
        },
        "animations": {
            "type": "array",
            "description": "An array of keyframe animations.",
            "items": {
                "$ref": "animation.schema.json"
            },
            "minItems": 1
        },
        "asset": {
            "allOf": [ { "$ref": "asset.schema.json" } ],
            "description": "Metadata about the glTF asset."
```