Package 'rgl2gltf'

August 19, 2022

Type Package

Title Read and Write '.gltf' and '.glb' Files

Version 1.0.3

Maintainer Duncan Murdoch <murdoch.duncan@gmail.com>

Description The 'glTF' file format is used to describe 3D models. This package provides read and write functions to work with it.

License GPL-2

Encoding UTF-8

Depends R (>= 3.5.0)

Imports jsonlite, rgl (>= 0.108.43), grDevices, R6, base64enc

URL https://github.com/dmurdoch/rgl2gltf

BugReports https://github.com/dmurdoch/rgl2gltf/issues

Suggests rmarkdown, knitr, misc3d, V8, manipulateWidget, png, jpeg

VignetteBuilder knitr, rmarkdown

RoxygenNote 7.2.1

NeedsCompilation yes

Author Duncan Murdoch [aut, cre], Morten S. Mikkelsen [cph]

Repository CRAN

Date/Publication 2022-08-19 21:40:02 UTC

R topics documented:

`as.gltf` *Produce glTF objects*

Description

The glTF file is the JSON part of a glTF representation of a 3D scene. This function creates the R structure corresponding to one, and writes the binary buffer file for it.

Usage

```
as.gltf(x, ...)

## Default S3 method:
as.gltf(x, y = NULL, z = NULL, vertices,
                            material = NULL,
                            normals = NULL,
                            texcoords = NULL,
                            points = NULL, segments = NULL,
                            triangles = NULL,
                            quads = NULL,
                            transform = diag(4),
                            extras = NULL,
                            ...,
                            rglscene = list(),
                            previous = Gltf$new(),
                            newScene = FALSE,
                            parentNode = NULL,
                            dir = tempdir(),
                            scale = c(1,1,1))

## S3 method for class 'rglscene'
as.gltf(x, ..., previous = Gltf$new(),
                            newScene = FALSE)
## S3 method for class 'mesh3d'
as.gltf(x, ...)
```

Arguments

x	An object to convert to a "gltf" object.
y, z	In the default method, combined with x to make coordinates. Any reasonable way of defining the coordinates is acceptable. See the function xyz.coords for details.
vertices	A 3 or 4 row matrix of Euclidean or homogeneous coordinates; takes precedence over x, y, z.
material	material properties for rendering
normals	normals at each vertex as a 3 or 4 **column** matrix
texcoords	texture coordinates at each vertex as a 2 column matrix
points	vector of indices of vertices to draw as points
segments	2 x n matrix of indices of vertices to draw as segments
triangles	3 x n matrix of indices of vertices to draw as triangles
quads	4 x n matrix of indices of vertices to draw as quads
transform	4 x 4 matrix associated with this object (e.g. a subscene)
extras	A list to attach as extras component to the result
...	Other parameters passed to the default method.
rglscene	The RGL scene this came from, e.g. to look up defaults
previous	Optionally a previously produced "gltf" object; the new geometry will be added to it.
newScene	logical; if TRUE, add a new scene to previous, otherwise try to add to the existing scene.
parentNode	If not NULL, add the new object as a child of this node, otherwise add to the first node of the default scene.
dir	Where to write the binary buffer file.
scale	Rescaling in the enclosing subscene.

Details

as.gltf is a generic function.

The method for "rglscene" objects can handle most objects produced by scene3d, but not all objects will be handled. In particular:

- Lights, text, bounding box decorations and backgrounds are saved in "extra" fields, so they can be read by **rgl2gltf**, but most other software will ignore them or only display some parts.
- Most material properties are also stored in "extra" fields.

There are methods for many individual types of "rglobject", but these are intended for internal use.

Value

A "gltf" object.

Author(s)

Duncan Murdoch

References

The specification of the glTF format: https://registry.khronos.org/glTF/specs/2.0/glTF-2.0.html

Examples

```
cube <- rgl::rotate3d(rgl::cube3d(col = "red"), -pi/10, 1,0,0)
gltf <- as.gltf(cube)
rgl::plot3d(gltf)
gltf$closeBuffers()
```

`as.rglscene.gltf` *Convert a glTF object to an rglscene or mesh3d object.*

Description

These methods convert a `"gltf"` object to a `"rglscene"` object, similar to what `scene3d` would produce, or a `"mesh3d"` object.

Usage

```
## S3 method for class 'gltf'
as.rglscene(x, scene = x$scene, nodes = NULL,
         useRGLinfo = TRUE, time = NULL, ani = 0, clone = TRUE,
         quick = FALSE, add = FALSE, ...)
## S3 method for class 'gltf'
as.mesh3d(x, ...)
```

Arguments

`x`	The `"gltf"` object to convert.
`scene`	Which scene to convert? If `NULL` (e.g. `x` doesn't define a default scene), scene 0 will be used.
`nodes`	Which nodes to convert? If `NULL`, all nodes used in the scene will be converted, otherwise only the listed ones and their children.
`useRGLinfo`	`"gltf"` objects contain RGL-specific information in "extra" fields. If `useRGLinfo` is `TRUE`, we use that information, otherwise include only what standard glTF viewers would display.
`time`	Set the "time" within an animation.
`ani`	Animation number to use.
`clone`	Whether to clone the `gltf` object. See the Details below.

quick	If TRUE, plot3d will work on the result, but it is not sufficiently complete to use as the scene in rglwidget.
add	If quick = FALSE, the scene will be plotted in an existing rgl scene, and both old and new parts will be returned.
...	as.mesh3d passes these arguments to as.rglscene.

Details

These functions need to modify the glTF object, caching some information to help with the conversion. By default they do this on a cloned copy of it, and the original is left unchanged. If the object has already been cloned this can be skipped by setting clone = FALSE.

Value

An "rglscene" object.

---

extractTexture *Extract a texture file from a glTF object*

Description

Extracts a texture from a glTF object, and writes it to a file.

Usage

```
extractTexture(gltf, index = 0,
               outfile = tempfile(),
               verbose = TRUE)
```

Arguments

gltf	The glTF object.
index	The texture number (starting from 0).
outfile	The filename to write to. If outfile has no file extension, one will be added based on the MIME type of the texture.
verbose	Whether to report on success.

Details

Since **rgl** doesn't support any texture format except PNG, this function will attempt to convert JPEG textures to PNG. To do that it needs to have the **jpeg** and **png** packages available.

Value

Returns the filename that was written, or NULL if the request failed.

If closeConnections = FALSE, the filename will have attribute "gltf" containing the gltf object which might now contain a new open connection.

If the texture has a recorded MIME type in gltf, that will be returned in attribute "mimeType".

findEntry *Find a component of a recursive object*

Description

findEntry searches recursive objects for components matching a condition. namePattern creates a test of whether the component name matches a pattern. hasClass creates a test of whether the component has a class.

Usage

```
findEntry(x, test, ..., path = c())
namePattern(pattern)
hasClass(class)
```

Arguments

x	The recursive object to search.
test	A test function. See Details below.
pattern	A regexp pattern to match the desired name(s).
class	A class name to search for.
...	Optional additional arguments to pass to the test function.
path	For internal use: names to prepend to the path.

Details

Utility/debugging functions to search a large recursive object for a particular component name or class.

The test function should have header function(name, value), and may include other arguments which will be taken from ....

Value

findEntry returns a list with one entry per hit. Each entry in the list is a named vector giving the path to the hit, numerically in the values, and as an R expression by concatenating the names. The test functions will be passed single names and values, and should return a single logical result.

Examples

```
x <- list( a = list( b = list(c(d="A", e="B"), 1L, 1:3)))
locations <- findEntry(x, namePattern("e"))
locations

#This shows how the result can be used:
x[[locations[[1]]]]
expr <- paste0(c("x", names(locations[[1]])), collapse = "")
```

```
expr
eval(parse(text=expr))

findEntry(x, hasClass("integer"))
```

`getTangents` *Use the MikkTSpace code to generate tangent vectors.*

Description

This function generates tangent vectors using the MikkTSpace code by Morten S. Mikkelsen.

Usage

```
getTangents(obj)
```

Arguments

`obj`	A triangles or quads object as returned by `rgl::scene3d()`.

Details

glTF files include normal textures, which require the tangent space to be specified: the normals at each vertex are supplemented with a tangent vector and a bitangent vector that is their cross product. The standard recommends that if the glTF file doesn't specify tangents, they should be generated using the MikkTSpace code.

Note that a comment in '`mikktspace.h`' indicates that indexing needs to be recalculated after computing the tangents, so this function works on unindexed inputs, and reapplies indexing at the end.

Value

A modified copy of the original object, adding a 4 column `tangents` entry. The order and number of indices may have changed.

Author(s)

Morten S. Mikkelsen, Johannes Kuehnel and Duncan Murdoch.

References

The MikkTSpace code was obtained from https://github.com/mmikk/MikkTSpace. The interface code is based on code from https://www.turais.de/using-mikktspace-in-your-project/ by Johannes Kuehnel.

Gltf *R6 Class for glTF file objects*

Description

The glTF file spec is described here: https://registry.khronos.org/glTF/specs/2.0/glTF-2.0.html. This object encapsulates most of the data from those files.

Super class

rgl::Buffer -> gltf

Public fields

scene The default scene number

Methods

Public methods:

- Gltf$new()
- Gltf$addAccessor()
- Gltf$getScene()
- Gltf$setScene()
- Gltf$addScene()
- Gltf$addToScene()
- Gltf$defaultScene()
- Gltf$getNode()
- Gltf$setNode()
- Gltf$addNode()
- Gltf$addChild()
- Gltf$setParents()
- Gltf$getSkin()
- Gltf$setSkin()
- Gltf$getJoint()
- Gltf$getInverseBindMatrices()
- Gltf$getForwardBindMatrices()
- Gltf$getCamera()
- Gltf$getExtras()
- Gltf$setExtras()
- Gltf$getMesh()
- Gltf$setMesh()
- Gltf$addMesh()
- Gltf$getMaterial()

- Gltf$getTexture()
- Gltf$getImage()
- Gltf$addMaterial()
- Gltf$addTexture()
- Gltf$addImage()
- Gltf$addSampler()
- Gltf$getMaterialNumber()
- Gltf$writeVectors()
- Gltf$makePrimitive()
- Gltf$getAsset()
- Gltf$setAsset()
- Gltf$getTransform()
- Gltf$getRglMaterial()
- Gltf$getAnimation()
- Gltf$setAnimation()
- Gltf$timerange()
- Gltf$initAnimation()
- Gltf$settime()
- Gltf$print()
- Gltf$listCount()
- Gltf$clone()

Method `new()`**:**

Usage:

```
Gltf$new(json = NULL, defaultbin = NULL)
```

Arguments:

`json` List read from glTF file.

`defaultbin` Optional external binary file.

Method `addAccessor()`**:** Write values to accessor, including `min` and `max`.

The glTF standard requires `min` and `max` values in accessors, whereas other uses of buffers may not. This function stores in the usual way using the `Buffer$addAccessor()` method, and then adds `min` and `max` values.

The standard also doesn't support signed 4 byte integers or double precision values, so we test for those here.

Usage:

```
Gltf$addAccessor(values, target = NULL, types = "anyGLTF", normalized = FALSE)
```

Arguments:

`values` Values to write.

`target` Optional target use for values.

`types` Allowed types (from names of `rgl::gltfTypes`), or `c("any", "anyGLTF")`).

`normalized` Are these normalized integer values?

Returns: New accessor number.

Method `getScene()`**:** Get scene object.

Usage:

```
Gltf$getScene(sc)
```

Arguments:

`sc` Scene number.

Returns: Scene object, documented here: https://registry.khronos.org/glTF/specs/2.0/glTF-2.0.html#reference-scene.

Method `setScene()`**:** Update scene record.

Usage:

```
Gltf$setScene(sc, scene)
```

Arguments:

`sc` Which scene to update.

`scene` New scene record.

Method `addScene()`**:** Add a scene object.

Usage:

```
Gltf$addScene()
```

Returns: Scene number.

Method `addToScene()`**:** Add node to scene.

Usage:

```
Gltf$addToScene(scene, node)
```

Arguments:

`scene` Scene number to modify.

`node` Node number(s) to add.

Method `defaultScene()`**:** Get default scene, creating it if necessary.

Usage:

```
Gltf$defaultScene()
```

Returns: Scene number.

Method `getNode()`**:** Get node object.

Usage:

```
Gltf$getNode(n)
```

Arguments:

`n` Node number.

Returns: Node object, documented here: https://registry.khronos.org/glTF/specs/2.0/glTF-2.0.html#reference-node.

Method `setNode()`**:** Set node object.

Usage:

```
Gltf$setNode(n, node)
```

Arguments:

`n` Node number.

`node` New node object.

Method `addNode()`**:** Add a node object.

Usage:

```
Gltf$addNode(mesh = NULL, matrix = NULL, extras = NULL)
```

Arguments:

`mesh` A mesh number.

`matrix` A matrix transformation for the node.

`extras` A list of extras, typically `rgl` objects.

Returns: Node number.

Method `addChild()`**:** Add node as child of another.

Usage:

```
Gltf$addChild(parent, node)
```

Arguments:

`parent` Node number to modify.

`node` Node number(s) to add as children.

Method `setParents()`**:** Set parent member for all nodes

Usage:

```
Gltf$setParents()
```

Method `getSkin()`**:** Get skin object.

Usage:

```
Gltf$getSkin(skin)
```

Arguments:

`skin` Skin number.

Returns: Skin object, documented here: https://registry.khronos.org/glTF/specs/2.0/glTF-2.0.html#reference-skin.

Method `setSkin()`**:** Set skin object.

Usage:

```
Gltf$setSkin(n, skin)
```

Arguments:

`n` Skin number.

`skin` New skin object.

Method `getJoint()`**:** Get joint node.

Usage:

```
Gltf$getJoint(skin, num)
```

Arguments:

`skin` Skin number.

`num` Joint number.

Returns: Node object

Method `getInverseBindMatrices()`**:** Get "inverse bind matrices".
These matrices undo the existing transformation before applying the skin transformations.

Usage:

```
Gltf$getInverseBindMatrices(skin)
```

Arguments:

`skin` Skin number.

Returns: A 4x4xn array of matrices, one per joint.

Method `getForwardBindMatrices()`**:** Get "forward bind matrices".
These matrices applying the skin transformations.

Usage:

```
Gltf$getForwardBindMatrices(skin)
```

Arguments:

`skin` Skin number.

Returns: A 4x4xn array of matrices, one per joint.

Method `getCamera()`**:** Get camera object.

Usage:

```
Gltf$getCamera(cam)
```

Arguments:

`cam` Camera number.

Returns: Camera object, documented here: https://registry.khronos.org/glTF/specs/2.0/glTF-2.0.html#reference-camera.

Method `getExtras()`**:** Get top-level extras list.

Usage:

```
Gltf$getExtras()
```

Returns: Extras list, including rgl objects.

Method `setExtras()`**:** Set extras list.

Usage:

```
Gltf$setExtras(extras)
```

Arguments:

`extras` New extras list.

Method `getMesh()`: Get mesh object.

Usage:

```
Gltf$getMesh(m)
```

Arguments:

`m` Mesh number.

Returns: Mesh object, documented here: https://registry.khronos.org/glTF/specs/2.0/glTF-2.0.html#reference-mesh.

Method `setMesh()`: Set mesh object.

Usage:

```
Gltf$setMesh(m, mesh)
```

Arguments:

`m` Mesh number.

`mesh` New mesh object

Method `addMesh()`: Add a mesh object.

Usage:

```
Gltf$addMesh(primitives)
```

Arguments:

`primitives` A list of primitive objects.

Returns: Mesh number.

Method `getMaterial()`: Get material object.

Usage:

```
Gltf$getMaterial(m)
```

Arguments:

`m` Material number.

Returns: Material object, documented here: https://registry.khronos.org/glTF/specs/2.0/glTF-2.0.html#reference-material.

Method `getTexture()`: Get texture object.

Usage:

```
Gltf$getTexture(tex)
```

Arguments:

`tex` Texture number.

Returns: Texture object, documented here: https://registry.khronos.org/glTF/specs/2.0/glTF-2.0.html#reference-texture.

Method `getImage()`: Get image object.

Usage:

```
Gltf$getImage(im)
```

Arguments:

`im` Image number.

Returns: Image object, documented here: https://registry.khronos.org/glTF/specs/2.0/glTF-2.0.html#reference-image.

Method `addMaterial()`**:** Construct and possibly add material.
This will return an existing material if possible.

Usage:

```
Gltf$addMaterial(mat, defaultMaterial = list())
```

Arguments:

`mat` An `rgl` material record.

`defaultMaterial` Default material properties.

Returns: Material number.

Method `addTexture()`**:** Add a texture.

Usage:

```
Gltf$addTexture(mat)
```

Arguments:

`mat` An `rgl` material record.

Returns: Texture number.

Method `addImage()`**:** Add an image for a texture.

Usage:

```
Gltf$addImage(mat)
```

Arguments:

`mat` An `rgl` material record.

Returns: Image number.

Method `addSampler()`**:** Add a sampler.

Usage:

```
Gltf$addSampler(mat)
```

Arguments:

`mat` An `rgl` material record.

Returns: Sampler number.

Method `getMaterialNumber()`**:** Add or return a material.

Usage:

```
Gltf$getMaterialNumber(material)
```

Arguments:

`material` A glTF material record.

Returns: Material number.

Method `writeVectors()`**:** Write data.

Usage:

```
Gltf$writeVectors(
  coords,
  target = targetArray,
  types = "anyGLTF",
  normalized = FALSE
)
```

Arguments:

`coords` Data to write, or `NULL`.

`target` Optional target use for data.

`types` A character vector of allowed types, or "any" or "anyGLTF"

`normalized` Are these integer values representing floats?

Returns: Accessor number, or `NULL`.

Method `makePrimitive()`**:** Create a primitive record.

Usage:

```
Gltf$makePrimitive(inds, mode = NULL, attributes = NULL, matnum = NULL)
```

Arguments:

`inds` Indices of vertices.

`mode` Mode of primitive.

`attributes` Primitive attributes.

`matnum` Material number.

Returns: Primitive record, documented here: https://registry.khronos.org/glTF/specs/2.0/glTF-2.0.html#reference-primitive.

Method `getAsset()`**:** Get asset list.

Usage:

```
Gltf$getAsset()
```

Returns: Asset object, documented here: https://registry.khronos.org/glTF/specs/2.0/glTF-2.0.html#reference-asset.

Method `setAsset()`**:** Set asset list.

Usage:

```
Gltf$setAsset(version, generator)
```

Arguments:

`version` Version number of glTF format.

`generator` Identifier of code generating it.

Method `getTransform()`**:** Get local transform.

Usage:

```
Gltf$getTransform(n)
```

Arguments:

`n` Node number.

Returns: 4x4 matrix of local transform.

Method `getRglMaterial()`**:** Reconstruct `rgl` material.

Usage:

```
Gltf$getRglMaterial(n)
```

Arguments:

`n` Material number.

Returns: `rgl` material record.

Method `getAnimation()`**:** Get animation.

Usage:

```
Gltf$getAnimation(ani)
```

Arguments:

`ani` Animation number

Returns: Animation object, documented here: https://registry.khronos.org/glTF/specs/2.0/glTF-2.0.html#reference-animation.

Method `setAnimation()`**:** Set animation.

Usage:

```
Gltf$setAnimation(ani, animation)
```

Arguments:

`ani` Animation number

`animation` New animation object

Method `timerange()`**:** Find time range of an animation

Usage:

```
Gltf$timerange(ani)
```

Arguments:

`ani` Animation number

Returns: Min and max times from the samplers in the animation.

Method `initAnimation()`**:** Initialize animation.
This builds all of the interpolation functions in the samplers.

Usage:

```
Gltf$initAnimation(ani)
```

Arguments:

`ani` Animation number

Returns: Modified animation object

Method `settime()`**:** Set time for an animation.
This evaluates all the interpolators and modifies self to reflect the specified time point.

Usage:

```
Gltf$settime(time, ani = 0)
```

Arguments:

`time` Time to set.

`ani` Animation number.

Returns: Vector of node numbers that were changed.

Method `print()`**:** Print `gltf` objects with various levels of detail.

Usage:

```
Gltf$print(verbose = FALSE, names = FALSE, showExtras = TRUE, ...)
```

Arguments:

`verbose` Logical indicator of verbose printing, or character vector of components to print verbosely.

`names` Print names for components.

`showExtras` Logical: show extra fields?

`...` Passed `...` .

Examples:

```
\donttest{
samples <- "https://raw.githubusercontent.com/KhronosGroup/glTF-Sample-Models/master/2.0"
gltf <- readGLB(paste0(samples, "/2CylinderEngine/glTF-Binary/2CylinderEngine.glb?raw=true"))
gltf$print(names = "meshes")
}
```

Method `listCount()`**:** Get number of items in private list.

Usage:

```
Gltf$listCount(list)
```

Arguments:

`list` Name of list to get.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Gltf$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Gltf$print`
## ------------------------------------------------


samples <- "https://raw.githubusercontent.com/KhronosGroup/glTF-Sample-Models/master/2.0"
gltf <- readGLB(paste0(samples, "/2CylinderEngine/glTF-Binary/2CylinderEngine.glb?raw=true"))
gltf$print(names = "meshes")
```

gltfWidget *Create a widget for a glTF display.*

Description

This creates a widget holding a glTF scene, with controls to animate it if it supports animation, and with shaders that implement normal textures.

Usage

```
gltfWidget(gltf,
           animation = 0,
           start = times[1], stop = times[2],
           times = gltf$timerange(animation),
           method = c("shader", "rigid"),
           add = FALSE, close = !add,
           verbose = FALSE,
           open3dParams = getr3dDefaults(),
           usePBR = hasPBRparams(gltf),
           PBRargs = list(), ...)
```

Arguments

gltf	A "gltf" object, e.g. produced by readglTF.
animation	Which animation to use? If no animation is present, this is ignored.
start, stop	The starting and stopping times for the animation. Ignored if no animation.
times	An alternate way to specify the times. Ignored if no animation.
method	The "shader" method installs a custom shader to implement the animation in GLSL, as intended for glTF. This is necessary for the normal texture support. The "rigid" method duplicates vertices as necessary so that all triangles remain rigid as the vertices move, and should match the corresponding method in playgltf.
add	Should the gltf object be added to an existing **rgl** scene, or should it open a new scene?
close	Should gltfWidget close the **rgl** scene after producing the widget?
verbose	Give some progress information.
open3dParams	A list to pass as the params argument to open3d.
usePBR	Whether to use physically based rendering methods. The default uses an internal function to determine if gltf actually contains PBR parameters.
PBRargs	A list containing optional arguments to the setPBRshaders function.
...	Additional parameters which will be passed to rgl::playwidget.

Details

See playgltf for a description of the method used for animation. The "fixed" method is fast, but doesn't do a good job on some animations.

If the gltf object doesn't contain any animations, or animation = NA, this will simply display it as a widget with no controls.

Value

A widget suitable for display or inclusion in an R Markdown document.

Note

Physically based rendering (PBR, controlled by usePBR) is only used with method = "shader".

If PBR is not used, the **V8** package is required so that the shaders can be modified.

Examples

```
if ((interactive() || rgl::in_pkgdown_example()) && requireNamespace("manipulateWidget")) {
  gltf <- readGLB(system.file("glb/RiggedSimple.glb", package = "rgl2gltf"))
  gltfWidget(gltf)
}
```

---

imports *Imports from rgl*

Description

These generic functions are imported from **rgl**.

Usage

```
plot3d(x, ...)
as.rglscene(x, ...)
as.mesh3d(x, ...)
```

Arguments

x	Objects to handle.
...	Additional arguments to pass to methods.

Details

The main documentation for these generics is in **rgl**: plot3d.

This package defines "gltf" methods for the generics. Documentation for as.rglscene.gltf is included.

matrixSequence *Print the sequence of matrices affecting an object.*

Description

This is used for debugging, to see how an object within nested **rgl** subscenes is affected by their user matrices.

Usage

```
matrixSequence(tag, scene = scene3d())
## S3 method for class 'matrixSequence'
print(x, n = 5, ...)
```

Arguments

tag	Which objects to report on?
scene	The **rgl** scene to work from.
x	Object to print.
n	In the print method, how many vertices and indices should be printed?
...	Ignored.

Value

A list containing records with entries as follows:

id	The object id
vertices	The object's vertices
indices	The object's indices
userMatrix	A list of user matrices affecting this object

modifyShaders *Edit shader code*

Description

This function applies specified edits to shader code to support new functions.

Usage

```
modifyShaders(shaders, mod, ...)
```

Arguments

shaders	A list of shaders, e.g. as produced by getShaders.
mod	Either a character value naming a built-in set of mods (currently just "skins"), or a list of mods as described in the Details section.
...	Optional arguments; all will be passed to any functions in mods.

Details

The mods argument is organized in a nested list.

1. The top level can contain vertexShader and fragmentShader components.
2. Each of those is a list of changes to apply to that shader.
3. Each change is a list with old and new entries.
4. The old entry is used as a "fixed" pattern to select one or more lines from the shader to be replaced. Multiple matches are allowed, but they shouldn't overlap. Changes will be applied in the order specified, so take care that a pattern doesn't match new text from an earlier change.
5. The new entry contains the replacement. It can either be a character vector or a function that returns a character vector. If it is a function, it will be passed the ... argument.

   See rgl2gltf:::shaderChanges for the built-in modifications.

Value

A new shader object incorporating the edits.

---

playgltf *Play an animated glTF object.*

Description

'.gltf' and '.glb' files can contain animation instructions. This function interprets them and plays them.

Usage

```
playgltf(gltf, animation = 0,
         start = times[1], stop = times[2],
         times = gltf$timerange(animation),
         method = c("rigid", "wholeScene", "partialScene"),
         speed = 1, by = NULL, verbose = FALSE, ...)
showNodes(gltf, animation = 0,
          start = times[1], stop = times[2],
          times = gltf$timerange(animation),
          speed = 1, by = NULL)
```

Arguments

gltf	A "gltf" object.
animation	Which animation from the object? An integer from 0 to the number of animations defined in gltf.
start, stop	Starting and stopping times.
times	An alternate way to specify start and stop.
method	Which drawing method to use? See details below.
speed, by	Control the updates; see details below.
verbose	Whether to print status updates.
...	Parameter settings to pass to plot3d.rglscene (and hence to open3d).

Details

glTF files are animated by time dependent changes to the transformations in their nodes. Those transformations correspond to RGL par3d("userMatrix") settings in subscenes and can sometimes be directly imported as such.

However, glTF files also support "skins", a computer graphics concept not supported in RGL. A skin is a way to say that different vertices of the same object (typically a triangle mesh) respond to different nodes. This allows shapes to be stretched, similar to skin on a moving body. RGL assumes that all graphics objects are rigid.

The playgltf function provides partial support for skins. Using the "wholeScene" method, it can modify the vertices of an entire scene and redraw the scene. Typically this is quite slow, and not very satisfactory. The "partialScene" method allows only the changed objects to be redrawn, which might help speed things up. Finally, the "rigid" method converts all polygons to rigid ones that are supported by **rgl**, so that motion is done by changes to the transformations. This is likely the fastest method, but for some animations the errors introduced by the conversion are unacceptably large.

The showNodes function displays each node number as text at the origin for that node. By default it plays the animation showing how the nodes move.

For both functions, the speed and by arguments specify the "times" at which the animation is drawn. If by is specified, then a frame is drawn at time start and subsequent frames increment the time by by. If it is NULL (the default), then the speed argument is used as a multiplier on the internal time (taken to be in seconds). For example, with the default speed = 1, the first frame will be drawn at time start, and when it is complete, the next one will be drawn according to how many seconds have passed in real time, etc.

Value

Called for the side effect of drawing the animation.

Examples

```
if (interactive() && !rgl::in_pkgdown_example()) {
# This example is fast enough using the "whole" method:
```

```
gltf1 <- readGLB(system.file("glb/RiggedSimple.glb", package = "rgl2gltf"))
playgltf(gltf1, start = 0, stop = 3, method = "whole")

# It looks terrible using the "rigid" method, because some triangles
# need to be deformed:

playgltf(gltf1, start = 0, stop = 3, method = "rigid")

# This example is too slow using anything but "rigid", but it's fine there:

samples <- "https://raw.githubusercontent.com/KhronosGroup/glTF-Sample-Models/master/2.0"
gltf2 <- readGLB(paste0(samples, "/BrainStem/glTF-Binary/BrainStem.glb?raw=true"))
playgltf(gltf2, start = 0, stop = 2, speed = 0.25, method = "rigid")
}
```

`readGLB` *Read a GLB file.*

Description

GLB is the self-contained binary format of glTF files. This function reads one, extracts the embedded files into the temporary directory, and returns a `"gltf"` object containing the information in R format.

Usage

```
readGLB(con, verbose = FALSE, ...)
```

Arguments

`con`	The connection or filename to read from.
`verbose`	Whether to report on the process.
`...`	Currently unused.

Value

An object of class `"gltf"`. This is an R version of the structure represented by a "glTF" file's JSON information.

Author(s)

Duncan Murdoch

References

The specification of the glTF format: https://registry.khronos.org/glTF/specs/2.0/glTF-2.0.html

See Also

readglTF

Examples

```
# This web page has lots of sample files

samples <- "https://raw.githubusercontent.com/KhronosGroup/glTF-Sample-Models/master/2.0"

# Get one of them:  an avocado

gltf <- readGLB(paste0(samples, "/Avocado/glTF-Binary/Avocado.glb?raw=true"))

if (interactive())
  rgl::plot3d(gltf)

if (rgl::in_pkgdown_example())
  gltfWidget(gltf)

gltf$closeBuffers()
```

---

readglTF *Read a glTF file*

Description

The glTF file is the JSON part of a glTF representation of a 3D scene. This function reads one and returns a "gltf" object containing the information in R format.

Typically most of the data for the scene is contained in other files named in this one, usually found in the same file folder.

Usage

```
readglTF(path, defaultbin = NULL, ...)
```

Arguments

path	The path to the file being read. R connections cannot be used.
defaultbin	The name of the default associated binary file, if it is not named in the JSON. This is typically used when the JSON has been extracted from a GLB file.
...	Not currently used.

Value

An object of class "gltf".

Author(s)

Duncan Murdoch

References

The specification of the glTF format: https://registry.khronos.org/glTF/specs/2.0/glTF-2.0.html

Examples

```
# This web page has lots of sample files

samples <- "https://raw.githubusercontent.com/KhronosGroup/glTF-Sample-Models/master/2.0"
filename <- tempfile(fileext = ".gltf")

# Get one of them:  a 2 cylinder engine.  We need both parts
# to be able to view it, though only the .gltf part is
# needed for readglTF()

download.file(paste0(samples, "/2CylinderEngine/glTF/2CylinderEngine.gltf"),
  destfile = filename)
download.file(paste0(samples, "/2CylinderEngine/glTF/2CylinderEngine0.bin?raw=true"),
  destfile = file.path(tempdir(), "2CylinderEngine0.bin"),
  mode = "wb")

gltf <- readglTF(filename)
gltf

# gltf files contain references to other files using
# relative paths, so we can only use them from their
# own directory
olddir <- setwd(dirname(filename))
rgl::plot3d(gltf)
setwd(olddir)
```

`setPBRshaders` *Set shaders for physically based rendering.*

Description

The glTF format is designed to hold objects which are intended for "physically based rendering", where the parameters of the object map to physical properties such as metallicity, roughness, etc. This function replaces the default **rgl** shaders with PBR shaders based on the reference implementation in https://github.com/KhronosGroup/glTF-Sample-Viewer/tree/88eda8c5358efe03128b72b6c5f5f6e5b6d023
shaders.

Usage

```
setPBRshaders(gltf, gltfMat, id,
              scene = scene3d(minimal = TRUE),
              useIBL = TRUE,
            brdfLUT = system.file("textures/brdfLUT.png", package = "rgl2gltf"),
             IBLspecular = system.file("textures/refmap.png", package = "rgl"),
          IBLdiffuse = system.file("textures/refmapblur.jpeg", package = "rgl2gltf"),
              debugBaseColor = 0,
              debugMetallic = 0,
              debugRoughness = 0,
              debugSpecularReflection = 0,
              debugGeometricOcclusion = 0,
              debugMicrofacetDistribution = 0,
              debugSpecContrib = 0,
              debugDiffuseContrib = 0,
              debugIBLDiffuse = 1,
              debugIBLSpecular = 1,
              defines = list(),
              uniforms = list(),
              attributes = list(),
              textures = list())
```

Arguments

`gltf, gltfMat`	A `"gltf"` object, and a material record from it.
`id, scene`	The **rgl** id of the corresponding object and the scene holding it.
`useIBL`	Whether to use image based lighting.
`brdfLUT`	The texture to use for the "bidirectional reflectance distribution function" lookup table.
`IBLspecular`	The texture to use for the "image based specular lighting".
`IBLdiffuse`	The texture to use for the "image based diffuse lighting".
`debugBaseColor, debugMetallic, debugRoughness, debugSpecularReflection, debugGeometricOcclusion, debug`	These are flags used for debugging. Setting one of these to 1 will display just that contribution to the rendering.
`debugIBLDiffuse, debugIBLSpecular`	Two more debugging settings. These should be set to non-negative values to control the contribution from each of those components to the image based lighting.
`defines, uniforms, attributes, textures`	Values to use in `setUserShaders` in addition to the ones determined by this function.

Details

rgl is designed to work with WebGL version 1, which doesn't support all of the features used in the reference shaders. In particular, no extensions are assumed, and the IBL textures are single 2D textures rather than cube maps.

Value

This function modifies the `id` object in `scene`, and returns the modified scene.

Author(s)

Duncan Murdoch for the adaptation to **rgl**, various others for the original shaders.

References

https://github.com/KhronosGroup/glTF-Sample-Viewer

See Also

`gltfWidget`

Examples

```
# This web page has lots of sample files

samples <- "https://raw.githubusercontent.com/KhronosGroup/glTF-Sample-Models/master/2.0"

# Get one of them:  a 2 cylinder engine

gltf <- readGLB(paste0(samples, "/NormalTangentTest/glTF-Binary/NormalTangentTest.glb?raw=true"))
gltfMat <- gltf$getMaterial(0)
scene <- as.rglscene(gltf)
id <- scene$objects[[1]]$id
scene <- setPBRshaders(gltf, gltfMat, id, scene)
cat(scene$objects[[1]]$userFragmentShader)
```

---

showTags *Debugging tool: show tags for objects in rgl scene*

Description

This function uses `text3d` to display the tags at the center of the objects they label.

Usage

```
showTags(tags = NULL, ids = NULL,
         subscenes = ids3d("subscene", subscene = 0)$id,
         depth_test = "always", ...)
```

Arguments

tags, ids	If non-NULL, display only these tags and ids.
subscenes	Which subscenes to examine.
depth_test	The "depth_test" material property to use on the displayed tags. The default will put the tag in front of everything, so it won't be obscured by other objects.
...	Other arguments to pass to text3d.

Details

If selected objects don't have tags, they will be labelled using their id value instead.

Value

The **rgl** ids of the text objects added (one text object per tagged object, with tags as names).

Examples

```
example("plot3d", package = "rgl")
showTags()
```

---

showtree *Show the tree structure of nodes in a glTF object or rglscene object.*

Description

glTF files contain nodes corresponding to objects and groups of objects in 3D scenes. They must be a tree or forest: a node can only have zero or one parent. This function displays that structure, or the similar structure for rglscene objects.

Usage

```
showtree(x, ...)
## S3 method for class 'gltf'
showtree(x, ...)
## S3 method for class 'rglscene'
showtree(x, transform = FALSE, ...)
```

Arguments

x	A "gltf" or "rglscene" object.
transform	Whether to show the transform associated with a subscene.
...	Additional arguments, currently unused.

writeglTF *Write a glTF or GLB file.*

Description

GLB is the self-contained binary format of glTF files. These functions write a "gltf" object to one of these formats.

Usage

```
writeglTF(x, path, bin = TRUE)
writeGLB(x, con)
```

Arguments

x	A "gltf" object, e.g. from as.gltf.
path	A filename in which to write the JSON part of the file.
bin	logical; whether or not to write the binary part of the object.
con	A filename or connection to which to write the GLB file.

Details

If bin = TRUE (the default), writeglTF will write the binary part of the object to one or more separate files with filename constructed by concatenating the main part of path, followed by a number and the extension '.bin'.

Value

writeglTF returns path invisibly. writeGLB returns NULL invisibly.

Author(s)

Duncan Murdoch

References

The specification of the glTF format: https://registry.khronos.org/glTF/specs/2.0/glTF-2.0.html

See Also

readglTF

Examples

```
filename <- tempfile(fileext = ".glb")
writeGLB(as.gltf(rgl::cube3d(col = "red")), filename)
```

Index