Package 'scimesh'

August 21, 2026

Type Package

Title Headless Publication-Quality 3D Mesh Rendering Engine

Version 0.3.4

Description A fast, GPU-free 3D software renderer written in modern C++17 with native R bindings. Renders triangle meshes to publication-quality images entirely on the CPU, requiring no display server or graphics hardware. Features multi-light Blinn-Phong shading, screen-space ambient occlusion, anti-aliasing, depth fog, transparency, wireframe rendering, texture mapping, and procedural geometry generation. Supports standard mesh file formats with PNG and PPM output. Works on high-performance computing clusters, headless servers, containers, and continuous integration pipelines, making it suitable for scientific visualization across neuro-imaging, molecular structures, and general 3D graphics.

License MIT + file LICENSE

URL https://github.com/dfsp-spirit/scimesh, https://dfsp-spirit.github.io/scimesh/

BugReports https://github.com/dfsp-spirit/scimesh/issues

Imports Rcpp (>= 1.0.0)

LinkingTo Rcpp

Suggests testthat (>= 3.0.0), png, freesurferformats, viridisLite, knitr, rmarkdown

VignetteBuilder knitr

SystemRequirements C++17

Config/testthat/edition 3

Encoding UTF-8

NeedsCompilation yes

Config/roxygen2/version 8.0.0

RoxygenNote 7.3.3

Author Tim Schäfer [aut, cre],
Martin Hořeňovský [ctb] (Author of Catch2
(cpp_tests/catch_amalgamated.{h,cpp})),
Christophe Riccio [ctb] (Author of GLM - OpenGL Mathematics
(src/third_party/glm/)),
Dimitri Diakopoulos [ctb] (Author of tinyply
(src/third_party/tinyply.{h,cpp})),
Syoyo Fujita [ctb] (Author of tinyobjloader
(src/third_party/tiny_obj_loader.h)),
Tim Schäfer [ctb] (Author of libfs (src/third_party/libfs.h)),
Sebastian Reiter [ctb] (Author of stl_reader
(src/third_party/stl_reader.h)),
Sean Barrett [ctb] (Author of stb libraries
(src/third_party/stb_image*.h))

Maintainer Tim Schäfer <ts+code@rcmd.org>

Repository CRAN

Date/Publication 2026-08-21 14:10:10 UTC

Contents

`apply_colormap` *Apply a colormap to numerical data*

Description

Maps numeric per-vertex (or per-element) data to RGBA colours using a colormap. Handles multi-dataset data (e.g., two brain hemispheres), NaN values, and optional outlier clipping via winsorizing.

Usage

```
apply_colormap(
  data,
  colormap = viridis_colormap(256L),
  limits = NULL,
  nan_color = c(0.5, 0.5, 0.5, 1),
  winsor_percentiles = NULL
)
```

Arguments

data	A numeric vector, or a list of numeric vectors for multi-dataset mapping (e.g., list(lh_data, rh_data)).
colormap	A colormap specification: a function f(n) returning n hex colour strings (e.g., viridis_colormap), a character vector of hex colours, or an Nx3/Nx4 matrix of RGBA values in [0,1]. Default is viridis_colormap(256L).
limits	How the data value range is determined. NULL (default) auto-detects from finite values after winsorizing. c(min, max) sets an explicit fixed range. "global" pools all datasets for a shared range. "each" uses independent per-dataset ranges.
nan_color	RGBA colour for NaN/NA values as a length-3 (RGB) or length-4 (RGBA) numeric vector in [0,1]. Default mid-grey.
winsor_percentiles	Optional c(lower, upper) percentiles for outlier clipping, e.g. c(0.02, 0.98). NULL disables winsorizing.

Value

If data is a single vector: an Nx4 numeric matrix of RGBA colours. If data is a list: a list of Nx4 matrices.

Attributes on the result provide metadata. Single-dataset: data_min, data_max, raw_min, raw_max, winsor_lo, winsor_hi, nan_count. Multi-dataset: pooled_data_min, pooled_data_max (use for a colourbar), data_ranges, winsor_cutoffs, nan_counts.

Examples

```
data <- c(1.2, 3.4, NA, 2.1, 5.0, 2.8)
colors <- apply_colormap(data)

noisy <- c(rnorm(95, mean = 50, sd = 10), 200, -50)
colors <- apply_colormap(noisy, winsor_percentiles = c(0.02, 0.98))

lh <- c(2.3, 2.1, NA, 3.4)
rh <- c(2.5, 2.0, NA, 3.1)

colors <- apply_colormap(list(lh, rh),
    colormap = viridis_colormap(256L),
    limits = "global",
    winsor_percentiles = c(0.02, 0.98),
    nan_color = c(1, 1, 1, 1))
```

---

camera *Create a camera specification*

Description

Defines a camera for rendering by specifying the eye position, look-at center, up vector, projection type, and field of view.

Usage

```
camera(
  eye,
  center,
  up = c(0, 1, 0),
  projection = c("perspective", "orthographic"),
  fov = 45
)
```

Arguments

`eye`	Numeric vector of length 3: camera position.
`center`	Numeric vector of length 3: point the camera looks at.
`up`	Numeric vector of length 3: camera up direction.
`projection`	Projection type: `"perspective"` (default) or `"orthographic"`.
`fov`	Field of view in degrees (perspective only).

Value

A camera list suitable for `render_mesh()` or `render_scene()`.

Examples

```
cam <- camera(eye = c(0, 0, 5), center = c(0, 0, 0))
cam$eye
```

`camera_auto` *Auto-frame a camera to fit a mesh or vertex set*

Description

Computes a camera position that frames the entire mesh in view. The camera looks along the given direction, positioned at a distance that ensures the mesh fits within the field of view.

Usage

```
camera_auto(
  mesh,
  direction = c(0, 0, -1),
  up = c(0, 1, 0),
  fov = 45,
  margin = 1.1,
  rgl_compat = FALSE,
  projection = c("perspective", "orthographic")
)
```

Arguments

mesh	Either an Nx3 numeric matrix of vertex positions, or a mesh descriptor list with a vertices component.
direction	The view direction as a length-3 vector. For example, c(0, 0, -1) looks along the negative Z axis. Ignored when rgl_compat = TRUE.
up	The up vector as a length-3 vector. Default c(0, 1, 0). Ignored when rgl_compat = TRUE.
fov	Field of view in degrees. Default 45° (30° when rgl_compat = TRUE).
margin	Extra margin factor (1.0 = tight fit, 1.1 = 10% margin).
rgl_compat	Logical. If TRUE, use rgl's camera defaults and bounding-sphere distance formula. Default FALSE.
projection	Projection type: "perspective" (default) or "orthographic". When orthographic, the camera distance is computed to tightly frame the mesh regardless of FOV.

Details

When rgl_compat = TRUE, the camera mimics rgl's default auto-framing behaviour: a 30° FOV, 15° elevation, and the distance is computed from the *bounding sphere* of the mesh (the half-diagonal of the axis-aligned bounding box), reproducing the formula distance = sphere_radius / sin(FOV/2) used by rgl.

Value

A camera list, with S3 class "scimesh_camera".

Examples

```
verts <- matrix(c(-1,-1,-1, 1,-1,-1, 1,1,-1, -1,1,-1,
                  -1,-1, 1, 1,-1, 1, 1,1, 1, -1,1, 1), ncol = 3, byrow = TRUE)
tris <- matrix(c(0L,3L,2L, 0L,2L,1L, 4L,5L,6L, 4L,6L,7L,
                 0L,1L,5L, 0L,5L,4L, 2L,3L,7L, 2L,7L,6L,
                 0L,4L,7L, 0L,7L,3L, 1L,2L,6L, 1L,6L,5L), ncol = 3, byrow = TRUE)
mesh <- list(vertices = verts, triangles = tris)
cam <- camera_auto(mesh, direction = c(1, 1, 1))
cam_rgl <- camera_auto(mesh, rgl_compat = TRUE)
```

---

camera_orbit *Orbit a camera around an axis*

Description

Rotates a camera's eye position and up vector around its center by a given angle about a rotation axis. Useful for generating turntable-style frame sequences.

Usage

```
camera_orbit(camera, axis = c(0, 0, 1), angle_degrees)
```

Arguments

camera	A camera list from camera() or camera_auto().
axis	Rotation axis as a length-3 vector. Default c(0, 0, 1) (Z axis).
angle_degrees	Rotation angle in degrees.

Value

A camera list with S3 class "scimesh_camera".

Examples

```
mesh <- generate_torus(c(0, 0, 0))
cam <- camera_auto(mesh, direction = c(1, 1, 1))
cam2 <- camera_orbit(cam, axis = c(0, 0, 1), angle_degrees = 90)
```

---

colorbar_horizontal *Generate a horizontal colorbar image*

Description

Creates a horizontal colorbar as a 4-channel RGBA array. The color strip and optional tick labels are rendered to PNG using base R graphics (headless-safe). Returns a 3D array suitable for image_to_array() or direct composition.

Usage

```
colorbar_horizontal(
  colormap,
  n_colors = 256L,
  width = 600L,
  height = 80L,
  ticks = NULL,
  tick_labels = NULL,
  data_range = c(0, 1),
  label_cex = 1,
  title = NULL,
  background = c(1, 1, 1, 1)
)
```

Arguments

colormap	A vector of colors or a function returning colors (e.g. grDevices::hcl.colors).
n_colors	Number of discrete color segments in the gradient.
width	Output width in pixels.
height	Output height in pixels.
ticks	Numeric vector of tick positions in data units (matching data_range). If NULL, ticks are computed automatically via pretty() restricted to the data range.
tick_labels	Character vector of tick labels. If NULL, defaults to formatted tick values.
data_range	The data range that ticks are specified in. Defaults to c(0, 1).
label_cex	Label size multiplier.
title	Optional title string drawn above the color strip (horizontal) or to the right (vertical).
background	Background RGBA color (0-1 scale).

Value

A 3D array of dimensions (height, width, 4) with values in [0, 1].

Examples

```
cbar <- colorbar_horizontal(viridis_colormap, data_range = c(-2, 3),
    title = "Value")
dim(cbar)  # height x width x 4
```

---

colorbar_vertical *Generate a vertical colorbar image*

Description

Creates a vertical colorbar as a 4-channel RGBA array.

Usage

```
colorbar_vertical(
  colormap,
  n_colors = 256L,
  width = 80L,
  height = 600L,
  ticks = NULL,
  tick_labels = NULL,
  data_range = c(0, 1),
  label_cex = 1,
  title = NULL,
  background = c(1, 1, 1, 1)
)
```

Arguments

colormap	A vector of colors or a function returning colors.
n_colors	Number of discrete color segments in the gradient.
width	Output width in pixels.
height	Output height in pixels.
ticks	Numeric vector of tick positions in data units. If NULL, ticks are computed automatically via pretty() restricted to the data range.
tick_labels	Character vector of tick labels.
data_range	The data range that ticks are specified in.
label_cex	Label size multiplier.
title	Optional title string drawn to the right of the color strip.
background	Background RGBA color (0-1 scale).

Value

A 3D array of dimensions (height, width, 4) with values in [0, 1].

Examples

```
cbar <- colorbar_vertical(viridis_colormap, data_range = c(0, 100),
    title = "Count")
dim(cbar)
```

---

compose_layout *Compose multiple images into a single figure*

Description

Arranges rendered images (from render_mesh() or render_scene()) into a grid layout and optionally appends a colorbar. All composition is done with pure R array operations.

Usage

```
compose_layout(
  images,
  nrow = NULL,
  ncol = NULL,
  colorbar = NULL,
  colorbar_height = 80L,
  colorbar_width = 80L,
  background = c(0, 0, 0, 0),
  colorbar_side = c("right", "left"),
  crop = FALSE
)
```

Arguments

images	A list of images, each a list with width, height, pixels as returned by render_mesh().
nrow	Number of rows in the grid layout.
ncol	Number of columns in the grid layout. If both nrow and ncol are NULL, a square-ish layout is chosen automatically.
colorbar	Optional colorbar array (from colorbar_horizontal() or colorbar_vertical()). Placed below if horizontal, to the right if vertical.
colorbar_height	Height of the colorbar row in pixels. Only used when appending a horizontal colorbar.
colorbar_width	Width of the colorbar column in pixels. Only used when appending a vertical colorbar.
background	Background RGBA color for padding (0-1 scale).
colorbar_side	For vertical colorbars, whether to place the bar on the "right" (default) or "left" of the brain images. Ignored for horizontal colorbars.
crop	Logical. If TRUE, transparent borders are cropped individually and images are padded to per-row height and per-column width for a tight layout with minimal white space. Default is FALSE (images must be same size).

Value

A list with width, height, pixels suitable for write_png() or image_to_array().

Examples

```
mesh1 <- generate_cuboid(c(-1.5, 0, 0), c(0.5, 0.5, 0.5), c(1, 0, 0, 1))
mesh2 <- generate_cuboid(c( 1.5, 0, 0), c(0.5, 0.5, 0.5), c(0, 0, 1, 1))
img1 <- render_mesh(mesh1$vertices, mesh1$triangles)
img2 <- render_mesh(mesh2$vertices, mesh2$triangles)
result <- compose_layout(list(img1, img2), nrow = 1L)
tmp_file <- tempfile(fileext = ".png")
write_png(result, tmp_file)
```

---

compute_vertex_normals

Compute per-vertex normals for a mesh

Description

Computes smooth vertex normals by averaging face normals. Returns the same mesh with a normals component (Nx3 numeric matrix). Useful for imported meshes that lack pre-computed normals.

Usage

```
compute_vertex_normals(mesh)
```

Arguments

mesh	A mesh descriptor list with vertices and triangles.

Value

The mesh with a normals component added.

Examples

```
mesh <- generate_cuboid(c(0, 0, 0), c(1, 1, 1))
mesh <- compute_vertex_normals(mesh)
nrow(mesh$normals)
```

---

diverging_colormap *Custom diverging colormap for neuroimaging*

Description

Returns a blue-white-red diverging colormap suitable for displaying signed morphometry data (e.g. cortical thickness Z-scores).

Usage

```
diverging_colormap(n)
```

Arguments

n	Number of colors.

Value

A character vector of hex color strings.

Examples

```
cols <- diverging_colormap(256)
plot(1:256, pch = 15, col = cols, cex = 2, axes = FALSE, xlab = "", ylab = "")
```

`generate_arrow` *Generate an arrow mesh*

Description

Creates a 3D arrow from `from` to `to`, with a cylindrical shaft and a conical head.

Usage

```
generate_arrow(
  from,
  to,
  shaft_radius = 0.1,
  head_radius = 0.3,
  head_length = 0.6,
  segments = 32,
  color = c(1, 1, 1, 1)
)
```

Arguments

`from`	Length-3 start point.
`to`	Length-3 end point (tip of the arrowhead).
`shaft_radius`	Radius of the shaft cylinder.
`head_radius`	Radius at the base of the conical head.
`head_length`	Length of the arrowhead.
`segments`	Subdivision count (default 32).
`color`	Length-4 RGBA colour.

Value

A mesh descriptor list.

Examples

```
mesh <- generate_arrow(c(0, 0, 0), c(0, 2, 0))
nrow(mesh$vertices)
```

generate_axes *Generate XYZ axis arrows as cylinder meshes*

Description

Creates three coloured arrow meshes (red X, green Y, blue Z) from a centre point.

Usage

```
generate_axes(center = c(0, 0, 0), size = 1, shaft_radius = 0.02)
```

Arguments

center	Length-3 vector: origin of the axes.
size	Length of each axis.
shaft_radius	Cylinder radius for axis shafts.

Value

A mesh descriptor list suitable for `render_mesh()` or inclusion in a scene list.

Examples

```
axes_mesh <- generate_axes(size = 2)
nrow(axes_mesh$vertices)
```

generate_bbox *Generate a wireframe bounding box mesh*

Description

Creates 12 edge segments around an axis-aligned bounding box.

Usage

```
generate_bbox(bbox, color = c(0, 0, 0, 1), radius = 0.01)
```

Arguments

bbox	A bounding box list from `mesh_bbox()`, or a mesh descriptor (in which case `mesh_bbox()` is called).
color	RGBA colour for the edges (length 4, 0-1 scale).
radius	Cylinder radius for the edges.

Value

A mesh descriptor list suitable for `render_mesh()` or inclusion in a scene list.

Examples

```
mesh <- generate_cuboid(c(0, 0, 0), c(1, 2, 3))
bbox_mesh <- generate_bbox(mesh)
nrow(bbox_mesh$vertices)
```

---

generate_cone *Generate a cone mesh*

Description

Creates a cone from `base` to `tip` with the given base `radius`, subdivided into `segments` around the axis. The base cap is included.

Usage

```
generate_cone(base, tip, radius = 0.5, segments = 32, color = c(1, 1, 1, 1))
```

Arguments

`base`	Length-3 vector: centre of the circular base.
`tip`	Length-3 vector: tip of the cone.
`radius`	Base radius.
`segments`	Subdivision count (default 32).
`color`	Length-4 RGBA colour.

Value

A mesh descriptor list.

Examples

```
mesh <- generate_cone(c(0, -1, 0), c(0, 1, 0), radius = 0.8)
nrow(mesh$vertices)
```

generate_cuboid *Generate a cuboid mesh*

Description

Creates an axis-aligned cuboid (box) centred at `center` with the given half-extents along each axis.

Usage

```
generate_cuboid(center, half_extents, color = c(0.7, 0.7, 0.7, 1))
```

Arguments

`center`	Length-3 vector: centre of the cuboid.
`half_extents`	Length-3 vector: half-width, half-height, half-depth.
`color`	Length-4 RGBA colour (0-1 scale).

Value

A mesh descriptor list.

Examples

```
mesh <- generate_cuboid(c(0, 0, 0), c(1, 2, 0.5))
nrow(mesh$vertices)
nrow(mesh$triangles)
```

generate_cylinder *Generate a cylinder mesh*

Description

Creates a cylinder from `start` to end with the given `radius`, subdivided into `segments` around the axis. Both end caps are included.

Usage

```
generate_cylinder(
  start,
  end,
  radius = 0.5,
  segments = 32,
  color = c(1, 1, 1, 1)
)
```

Arguments

start	Length-3 vector: cylinder start point.
end	Length-3 vector: cylinder end point.
radius	Cylinder radius.
segments	Subdivision count (default 32).
color	Length-4 RGBA colour.

Value

A mesh descriptor list.

Examples

```
mesh <- generate_cylinder(c(0, -1, 0), c(0, 1, 0), radius = 0.5)
nrow(mesh$vertices)
```

---

generate_plane *Generate a planar quad mesh*

Description

Creates a flat rectangular plane centred at `center` and oriented perpendicular to `normal`.

Usage

```
generate_plane(
  center = c(0, 0, 0),
  normal = c(0, 1, 0),
  half_size_x = 1,
  half_size_y = 1,
  color = c(0.7, 0.7, 0.7, 1)
)
```

Arguments

center	Length-3 vector: centre of the plane.
normal	Length-3 vector: surface normal.
half_size_x	Half-extent along the first tangent axis.
half_size_y	Half-extent along the second tangent axis.
color	Length-4 RGBA colour.

Value

A mesh descriptor list.

Examples

```
mesh <- generate_plane(c(0, 0, 0), normal = c(0, 1, 0),
                       half_size_x = 2, half_size_y = 1)
nrow(mesh$vertices)
```

`generate_pyramid` *Generate a square pyramid mesh*

Description

Creates a pyramid with a square base centred at `base_center` in the XZ plane, with the apex above it along Y.

Usage

```
generate_pyramid(
  base_center,
  apex,
  half_width = 1,
  color = c(0.7, 0.7, 0.7, 1)
)
```

Arguments

`base_center`	Length-3 vector: centre of the square base.
`apex`	Length-3 vector: position of the tip.
`half_width`	Half-width of the square base.
`color`	Length-4 RGBA colour.

Value

A mesh descriptor list.

Examples

```
mesh <- generate_pyramid(c(0, 0, 0), c(0, 2, 0), half_width = 1)
mesh$vertices
```

generate_sphere *Generate a sphere mesh*

Description

Creates a UV sphere centred at `center` with the given `radius`. The sphere is subdivided into `segments` rings and segments per ring.

Usage

```
generate_sphere(center, radius = 1, segments = 32, color = c(1, 1, 1, 1))
```

Arguments

`center`	Length-3 vector: sphere centre.
`radius`	Sphere radius.
`segments`	Subdivision count (default 32).
`color`	Length-4 RGBA colour.

Value

A mesh descriptor list.

Examples

```
mesh <- generate_sphere(c(0, 0, 0), radius = 1.5, segments = 32)
nrow(mesh$vertices)
```

generate_tetrahedron *Generate a tetrahedron mesh*

Description

Creates a tetrahedron (triangular pyramid) from four arbitrary 3D points.

Usage

```
generate_tetrahedron(p0, p1, p2, p3, color = c(0.7, 0.7, 0.7, 1))
```

Arguments

`p0`	Length-3 vector: first vertex.
`p1`	Length-3 vector: second vertex.
`p2`	Length-3 vector: third vertex.
`p3`	Length-3 vector: fourth vertex.
`color`	Length-4 RGBA colour.

Value

A mesh descriptor list.

Examples

```
mesh <- generate_tetrahedron(
  c(0, 0, 0), c(1, 0, 0),
  c(0.5, 1, 0), c(0.5, 0.5, 1))
nrow(mesh$vertices)
```

---

generate_torus *Generate a torus mesh*

Description

Creates a torus (donut shape) centred at `center`, lying in the XZ plane.

Usage

```
generate_torus(
  center = c(0, 0, 0),
  major_radius = 1,
  minor_radius = 0.3,
  major_segments = 32,
  minor_segments = 16,
  color = c(0.7, 0.7, 0.7, 1)
)
```

Arguments

`center`	Length-3 vector: centre of the torus.
`major_radius`	Radius of the ring (tube path).
`minor_radius`	Radius of the tube cross-section.
`major_segments`	Number of segments around the ring.
`minor_segments`	Number of segments around the tube.
`color`	Length-4 RGBA colour.

Value

A mesh descriptor list.

Examples

```
mesh <- generate_torus(major_radius = 2, minor_radius = 0.5)
nrow(mesh$vertices)
```

image_apply_contrast *Apply contrast adjustment to an image*

Description

Applies a contrast stretch (S-curve) to the RGB channels of a rendered image. Formula: `(value - 0.5) * contrast + 0.5`, clamped to `[0, 1]`. The default 1.0 means no change. Values > 1.0 produce darker darks and lighter highlights.

Usage

```
image_apply_contrast(image, contrast = 1)
```

Arguments

`image`	An image list returned by `render_mesh()` or `render_scene()`.
`contrast`	Contrast multiplier. Default 1.0 (no change). Typical values: 1.1–1.2 for subtle S-curve, 1.5 for strong.

Value

A new image list with contrast-adjusted pixel data.

Examples

```
cube <- generate_cuboid(c(0, 0, 0), c(1, 1, 1))
img <- render_mesh(cube$vertices, cube$triangles)
img <- image_apply_contrast(img, contrast = 1.1)
```

image_crop *Crop an image to a rectangular region*

Description

Crop an image to a rectangular region

Usage

```
image_crop(image, x, y, w, h)
```

Arguments

image	An image list returned by render_mesh() or similar.
x	Left edge of the crop region (0-based pixel coordinate).
y	Top edge of the crop region (0-based pixel coordinate).
w	Crop width in pixels.
h	Crop height in pixels.

Value

A new image list with the cropped dimensions.

Examples

```
cube <- generate_cuboid(c(0, 0, 0), c(1, 1, 1))
img <- render_mesh(cube$vertices, cube$triangles)
img <- image_crop(img, 100, 50, 400, 300)
```

---

image_crop_to_content *Crop an image to its content bounding box*

Description

Removes background-coloured margin from the specified edges of the image. The first non-background pixel found on each edge defines the crop boundary.

Usage

```
image_crop_to_content(image, direction, background)
```

Arguments

image	An image list.
direction	One of "left", "right", "horizontal" (both left and right), "top", "bottom", "vertical" (both top and bottom), or "all" (all four sides).
background	Numeric vector of length 4 with RGBA values in [0, 1] defining the background colour to crop away.

Value

A new image list with cropped dimensions.

Examples

```
cube <- generate_cuboid(c(0, 0, 0), c(1, 1, 1))
img <- render_mesh(cube$vertices, cube$triangles,
                   options = render_options(background_color = c(0, 0, 0, 0)))
img <- image_crop_to_content(img, "all", c(0, 0, 0, 0))
```

`image_grow` *Grow an image by adding padding*

Description

Expands the canvas by adding pixel rows/columns filled with a background colour.

Usage

```
image_grow(image, top, bottom, left, right, background)
```

Arguments

`image`	An image list.
`top`	Number of pixel rows to add above.
`bottom`	Number of pixel rows to add below.
`left`	Number of pixel columns to add to the left.
`right`	Number of pixel columns to add to the right.
`background`	Numeric vector of length 4 with RGBA values in `[0, 1]`.

Value

A new image list with the expanded dimensions.

Examples

```
cube <- generate_cuboid(c(0, 0, 0), c(1, 1, 1))
img <- render_mesh(cube$vertices, cube$triangles)
img <- image_grow(img, 10, 10, 20, 20, c(1, 1, 1, 1))
```

image_merge *Merge two images side by side or stacked*

Description

Merges another image into this one at the specified edge. For left/right merging, the heights must match. For top/bottom, the widths must match.

Usage

```
image_merge(image, other, direction)
```

Arguments

image	An image list.
other	Another image list.
direction	One of "left", "right", "top", "bottom".

Value

A new image list with the merged dimensions.

Examples

```
cube <- generate_cuboid(c(0, 0, 0), c(1, 1, 1))
sphere <- generate_sphere(c(0, 0, 0), radius = 0.5)
left  <- render_mesh(sphere$vertices, sphere$triangles)
right <- render_mesh(cube$vertices, cube$triangles)
merged <- image_merge(left, right, "right")
```

image_rotate_90 *Rotate an image by 90 degrees*

Description

Rotate an image by 90 degrees

Usage

```
image_rotate_90(image, clockwise = TRUE)
```

Arguments

image	An image list.
clockwise	Logical, if TRUE (default) rotates clockwise, otherwise counter-clockwise.

Value

A new image list with width and height swapped.

`image_scale` *Scale an image (nearest-neighbour)*

Description

Resizes the image to the given dimensions using nearest-neighbour interpolation.

Usage

```
image_scale(image, new_width, new_height)
```

Arguments

`image`	An image list.
`new_width`	Target width in pixels.
`new_height`	Target height in pixels.

Value

A new image list with the new dimensions.

`image_to_array` *Convert a rendered image to an RGBA array*

Description

Converts the output of `render_mesh()` or `render_scene()` into a 3-dimensional R array of dimensions (height x width x 4) with RGBA channels.

Usage

```
image_to_array(image)
```

Arguments

`image`	An image list returned by `render_mesh()` or `render_scene()`.

Value

A 3D array of dimensions (height, width, 4) with values in `[0, 1]`.

Examples

```
mesh <- generate_cuboid(c(0, 0, 0), c(1, 1, 1))
img <- render_mesh(mesh$vertices, mesh$triangles)
arr <- image_to_array(img)
dim(arr)  # height x width x 4
```

`mesh_bbox` *Compute the axis-aligned bounding box of a mesh*

Description

Compute the axis-aligned bounding box of a mesh

Usage

```
mesh_bbox(mesh)
```

Arguments

`mesh`	A mesh descriptor list with `vertices`.

Value

A list with `min` and `max` (each length-3 numeric).

Examples

```
mesh <- generate_cuboid(c(0, 0, 0), c(1, 2, 3))
bb <- mesh_bbox(mesh)
bb$min
bb$max
```

`mesh_from_rgl` *Convert an rgl tmesh3d to scimesh mesh format*

Description

Extracts vertices and triangle indices from an rgl `tmesh3d` object into the format expected by `render_mesh()`. Does not require the `rgl` package – any list with components `vb` (4xN homogeneous coordinates) and `it` (3xM index matrix) works.

Usage

```
mesh_from_rgl(tmesh)
```

Arguments

tmesh	A list with components vb and it, as produced by rgl::tmesh3d().

Value

A mesh descriptor list with vertices (Nx3) and triangles (Mx3, 1-based indices).

Examples

```
fake <- list(vb = rbind(0:3, 0:3, 0:3, rep(1, 4)),
             it = matrix(1:6, nrow = 3))
m <- mesh_from_rgl(fake)
m$vertices
m$triangles
```

---

mesh_to_rgl *Convert a scimesh mesh to rgl tmesh3d format*

Description

Builds an rgl-compatible triangular mesh from a scimesh mesh descriptor so that the result can be used with rgl::shade3d() or other rgl functions.

Usage

```
mesh_to_rgl(mesh, color = NULL, face_color = NULL)
```

Arguments

mesh	A scimesh mesh descriptor list with vertices (Nx3 matrix) and triangles (Mx3 integer matrix, 1-based).
color	Optional per-vertex colour, either a single length-4 RGBA vector (applied to all vertices) or an Nx4 matrix.
face_color	Optional per-face colour (Mx4 matrix).

Value

A list with components vb (4xN homogeneous coordinates), it (3xM 1-based index matrix), and optionally normals and mat (material), suitable for use with rgl's tmesh3d() and shade3d().

Examples

```
mesh <- generate_cuboid(c(0, 0, 0), c(1, 1, 1))
rgl_mesh <- mesh_to_rgl(mesh)
str(rgl_mesh)
```

read_obj *Read a Wavefront OBJ file*

Description

Reads a Wavefront OBJ file (with optional UV coordinates and normals) and returns a scimesh mesh descriptor list with `vertices`, `triangles`, and optionally `uv` and `normals`.

Usage

```
read_obj(path)
```

Arguments

`path`	Path to the OBJ file.

Value

A mesh descriptor list.

Examples

```
## Not run:
mesh <- read_obj("model.obj")
nrow(mesh$vertices)

## End(Not run)
```

read_ply *Read a Stanford PLY file*

Description

Reads a PLY file (ASCII or binary) with optional per-vertex colors and returns a scimesh mesh descriptor list with `vertices`, `triangles`, and optionally `colors`.

Usage

```
read_ply(path)
```

Arguments

`path`	Path to the PLY file.

Value

A mesh descriptor list.

Examples

```
## Not run:
mesh <- read_ply("model.ply")
nrow(mesh$vertices)

## End(Not run)
```

`read_stl` *Read an STL file*

Description

Reads an ASCII or binary STL file and returns a scimesh mesh descriptor list with `vertices`, `triangles`, and `normals`.

Usage

```
read_stl(path)
```

Arguments

`path`	Path to the STL file.

Value

A mesh descriptor list.

Examples

```
## Not run:
mesh <- read_stl("model.stl")
nrow(mesh$vertices)

## End(Not run)
```

render_lines *Render line segments as thin cylinders*

Description

Generates a merged cylinder mesh from start/end point pairs, radii, and colors, then renders it.

Usage

```
render_lines(
  from,
  to,
  radii = 0.1,
  colors,
  camera,
  options = render_options(),
  segments = 12L
)
```

Arguments

from	Nx3 numeric matrix of segment start points.
to	Nx3 numeric matrix of segment end points.
radii	Numeric vector of cylinder radii (length N, or 1 recycled to N).
colors	Nx4 numeric matrix of RGBA colours, or a single colour recycled to N.
camera	A camera list.
options	Render options.
segments	Number of sides around the cylinder (default 12).

Value

An image list.

Examples

```
from <- matrix(c(0, 0, 0, 1, 1, 1), ncol = 3, byrow = TRUE)
to   <- matrix(c(3, 0, 0, 0, 3, 0), ncol = 3, byrow = TRUE)
img <- render_lines(from, to, radii = 0.05,
                    colors = c(0, 0, 1, 1),
                    camera = camera_auto(rbind(from, to)))
tmp_file <- tempfile(fileext = ".png")
write_png(img, tmp_file)
```

render_mesh *Render a 3D mesh to an image*

Description

Renders a single mesh using the scimesh software renderer. The mesh can be specified either as separate `vertices`/`triangles` matrices, as a scimesh mesh descriptor list, or as an rgl `tmesh3d`-style list (with `vb`/`it` components). rgl meshes are transparently converted via `mesh_from_rgl()`.

Usage

```
render_mesh(
  vertices,
  triangles = NULL,
  colors = NULL,
  face_colors = NULL,
  normals = NULL,
  uv = NULL,
  texture = NULL,
  camera = NULL,
  options = render_options()
)
```

Arguments

`vertices`	Either an Nx3 numeric matrix of vertex positions, or a scimesh mesh descriptor list (with `vertices` and `triangles` components), or an rgl-style list (with `vb` and `it` components).
`triangles`	Mx3 integer matrix of triangle indices (1-based). Ignored when `vertices` is a list.
`colors`	Optional Nx4 numeric matrix of RGBA vertex colors (0-1). Use `face_colors` (Mx4) for per-triangle colours instead.
`face_colors`	Optional Mx4 numeric matrix of per-face RGBA colors, one row per triangle. When present, all three vertices of a triangle use the same colour. Takes precedence over vertex `colors`.
`normals`	Optional Nx3 numeric matrix of vertex normals.
`uv`	Optional Nx2 numeric matrix of texture coordinates (0-1).
`texture`	Optional texture image as a 3D array (H x W x 3 or 4) with values in `[0, 1]`, e.g. from `png::readPNG()`.
`camera`	A camera list from `camera()` or `camera_auto()`.
`options`	A render options list from `render_options()`.

Value

A list with components `width`, `height`, and `pixels` (raw vector of RGBA values).

Examples

```
# Render a simple colored triangle
verts <- matrix(c(0, 0, 0,  1, 0, 0,  0.5, 1, 0), ncol = 3, byrow = TRUE)
tris  <- matrix(1L, nrow = 1, ncol = 3)
cols  <- matrix(c(1, 0, 0, 1,  0, 1, 0, 1,  0, 0, 1, 1), ncol = 4, byrow = TRUE)
img <- render_mesh(verts, tris, colors = cols)
tmp_file <- tempfile(fileext = ".png")
write_png(img, tmp_file)

# Render from a mesh descriptor list (scimesh format)
mesh_desc <- list(vertices = verts, triangles = tris, colors = cols)
img <- render_mesh(mesh_desc)
```

---

render_options *Create render options*

Description

Create render options

Usage

```
render_options(
  width = 800L,
  height = 600L,
  shading = c("smooth", "flat"),
  backface_culling = TRUE,
  background_color = c(1, 1, 1, 1),
  default_color = c(0.7, 0.7, 0.7, 1),
  invert_normals = FALSE,
  wireframe = FALSE,
  wireframe_color = c(0, 0, 0, 1),
  projection = c("perspective", "orthographic"),
  specular_color = c(0, 0, 0, 0),
  shininess = 0,
  lights = NULL,
  ambient = 0.3,
  contrast = 1,
  fog_enabled = FALSE,
  fog_start = 0,
  fog_end = 1,
  fog_color = c(0, 0, 0, 0),
  threads = 0L,
  clip_planes = NULL,
  ssao_enabled = FALSE,
  ssao_radius = 16,
```

```
  ssao_intensity = 0.8,
  aa_samples = 1L
)
```

Arguments

width	Output image width in pixels.
height	Output image height in pixels.
shading	Shading mode: "smooth" or "flat".
backface_culling	Whether to cull back-facing triangles.
background_color	Background RGBA color as numeric vector of length 4 (values 0-1).
default_color	Default vertex color when no colors are provided.
invert_normals	Whether to invert surface normals.
wireframe	Whether to render in wireframe mode.
wireframe_color	RGBA color for wireframe edges (0-1 scale). Default c(0, 0, 0, 1) (black).
projection	Projection type: "perspective" (default) or "orthographic". Orthographic gives a parallel projection (no perspective foreshortening), matching rgl's view3d(fov=0) convention.
specular_color	Specular highlight color (0-1 scale). When shininess > 0, a Blinn-Phong highlight in this colour is added where the surface faces the camera. Default c(0, 0, 0, 0) (off).
shininess	Specular exponent controlling highlight sharpness. Higher values produce a tighter spot. Typical values: 32 (soft plastic), 64 (shiny), 128 (glass). Default 0 (off).
lights	A list of light descriptors, each a list with position (length-3 direction vector or point position), color (length-4 RGBA, 0-1 scale), intensity (numeric, default 1), and directional (logical, default TRUE). When empty or NULL, a single headlight at c(0, 0, 1) is used (the original behaviour).
ambient	Ambient light contribution (0-1). Default 0.3.
contrast	Contrast adjustment applied after shading, before uint8_t conversion. Default 1.0 (no change). Values > 1.0 produce darker darks and lighter highlights (S-curve). Formula: (value - 0.5) * contrast + 0.5, clamped to [0, 1].
fog_enabled	Enable depth cueing (fog). Default FALSE.
fog_start	Z-depth where fog begins (0 = near plane, 1 = far plane). Default 0.
fog_end	Z-depth where fog is fully opaque. Default 1.
fog_color	RGBA fog colour (0-1 scale). Defaults to background_color.
threads	Number of render threads. 0 = auto-detect (use all cores), 1 = single-threaded (deterministic). Default 0. Requires OpenMP at compile time.
clip_planes	A list of clip plane descriptors, each a list with normal (length-3 vector) and offset (numeric). Points satisfying dot(normal, position) + offset >= 0 are kept. Default NULL (no clipping).

ssao_enabled	Enable screen-space ambient occlusion. Default FALSE.
ssao_radius	Screen-space sample radius in pixels. Default 16.
ssao_intensity	Occlusion strength (0-1). Default 0.8.
aa_samples	Anti-aliasing supersampling factor. Renders internally at width * aa_samples x height * aa_samples, then downsamples to the requested size via box averaging. Default 1 (no AA), 2 for 2x2 SSAA, 4 for 4x4.

Value

A render options list for use with render_mesh() or render_scene().

Examples

```
# Default options
opts <- render_options()

# High-resolution with anti-aliasing and specular highlights
opts <- render_options(width = 1200, height = 900,
    aa_samples = 2L,
    specular_color = c(0.4, 0.4, 0.4, 1),
    shininess = 64)

# Wireframe with transparent background
opts <- render_options(wireframe = TRUE,
    wireframe_color = c(0, 0, 0, 1),
    background_color = c(0, 0, 0, 0))
```

---

render_points *Render screen-space point primitives*

Description

Renders points as fixed-size filled circles in screen space with depth testing. Unlike render_spheres(), point size is measured in pixels and does not change with camera distance.

Usage

```
render_points(
  positions,
  colors,
  radius = 3,
  camera = camera_auto(positions),
  options = render_options()
)
```

Arguments

positions	Nx3 numeric matrix of point positions.
colors	Nx4 numeric matrix of RGBA colours (0-1 scale).
radius	Point radius in pixels.
camera	A camera list.
options	Render options.

Value

An image list.

Examples

```
pts <- matrix(c(0, 1, 2, 0, 1, 2, 0, 0, 0),
 ncol = 3)
colors = matrix(c(0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1), ncol = 4)
img <- render_points(pts, colors = colors, radius = 5)
tmp_file <- tempfile(fileext = ".png")
write_png(img, tmp_file)
```

---

render_scene *Render multiple meshes to an image*

Description

Renders a list of meshes as a single scene using the scimesh software renderer. Each element can be a scimesh mesh descriptor or an rgl-style mesh (with vb/it); rgl meshes are transparently converted.

Usage

```
render_scene(meshes, camera = NULL, options = NULL)
```

Arguments

meshes	Either a scimesh_scene object (see scene()), a list of mesh descriptors, or a list of scene nodes. Each mesh descriptor is a list with components vertices (Nx3 matrix), triangles (Mx3 integer matrix), and optionally colors, face_colors, normals, and default_color. Elements may also be rgl-style lists (with vb and it), which are converted automatically.
camera	A camera list from camera() or camera_auto(). Ignored (falls back to the scene's camera) when meshes is a scimesh_scene and camera is NULL.
options	A render options list from render_options(). Defaults to the scene's options (if any) or render_options().

Value

A list with components `width`, `height`, and `pixels` (raw vector of RGBA values).

Examples

```
# Render two cubes side by side
cube1 <- generate_cuboid(c(-1.5, 0, 0), c(0.8, 0.8, 0.8), c(1, 0, 0, 1))
cube2 <- generate_cuboid(c( 1.5, 0, 0), c(0.8, 0.8, 0.8), c(0, 0, 1, 1))
cam <- camera_auto(list(cube1, cube2), direction = c(1, 1, 1))
img <- render_scene(list(cube1, cube2), cam,
    render_options(width = 800, height = 600, background_color = c(1, 1, 1, 1)))
tmp_file <- tempfile(fileext = ".png")
write_png(img, tmp_file)

# Render multiple meshes together
scimesh_cube <- generate_cuboid(c(-1, 0, 0), c(0.5, 0.5, 0.5))
sphere <- generate_sphere(c(1, 0, 0), radius = 0.5, color = c(0, 1, 0, 1))
cam <- camera_auto(list(scimesh_cube, sphere), direction = c(1, 1, 1))
img <- render_scene(list(scimesh_cube, sphere), cam,
    render_options(width = 400, height = 300, background_color = c(1, 1, 1, 1)))
```

---

render_spheres *Render multiple spheres from point data*

Description

Generates a merged sphere mesh from a set of center points, radii, and colors, then renders it with the given camera and options.

Usage

```
render_spheres(
  centers,
  radii,
  colors,
  camera,
  options = render_options(),
  segments = 16L
)
```

Arguments

`centers`	Nx3 numeric matrix of sphere centre coordinates.
`radii`	Numeric vector of sphere radii (length N, or 1 recycled to N).
`colors`	Nx4 numeric matrix of RGBA colours (0-1 scale), or a single colour recycled to N.

camera	A camera list from camera() or camera_auto().
options	Render options from render_options().
segments	Number of latitude/longitude segments per sphere (default 16).

Value

An image list with width, height, pixels.

Examples

```
centers <- matrix(c(0, 2, 4, 0, 0, 0, 0, 0, 0), ncol = 3)
img <- render_spheres(centers, radii = 0.5,
                      colors = c(1, 0, 0, 1),
                      camera = camera_auto(centers))
tmp_file <- tempfile(fileext = ".png")
write_png(img, tmp_file)
```

---

render_triangles *Render raw triangles without index buffer*

Description

Renders triangle geometry where positions and colours are given as flat arrays with 3 vertices per triangle (no index buffer). Useful for voxel renderings, misc3d isosurfaces, and other dynamically generated geometry.

Usage

```
render_triangles(positions, colors, camera, options = render_options())
```

Arguments

positions	Nx3 numeric matrix of vertex positions, where N is a multiple of 3 (3 per triangle).
colors	Nx4 numeric matrix of RGBA colours (0-1 scale).
camera	A camera list from camera() or camera_auto().
options	Render options from render_options().

Value

An image list with width, height, pixels.

Examples

```
# Render a single red triangle from raw vertices
positions <- matrix(c(0, 0, 0, 1, 0, 0, 0.5, 1, 0), ncol = 3, byrow = TRUE)
colors <- matrix(c(1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1), ncol = 4, byrow = TRUE)
cam <- camera_auto(positions)
img <- render_triangles(positions, colors, cam)
```

---

rotate_mesh *Rotate a mesh around an axis*

Description

Rotate a mesh around an axis

Usage

```
rotate_mesh(mesh, angle_rad, axis = c(0, 0, 1))
```

Arguments

mesh	A mesh descriptor list.
angle_rad	Rotation angle in radians.
axis	Length-3 numeric vector defining the rotation axis.

Value

A new mesh descriptor list with rotated vertices.

Examples

```
mesh <- generate_cuboid(c(0, 0, 0), c(1, 1, 1))
rotated <- rotate_mesh(mesh, pi / 4, axis = c(0, 1, 0))
rotated$vertices[1, ]
```

scale_mesh *Scale a mesh uniformly or per-axis*

Description

Scale a mesh uniformly or per-axis

Usage

```
scale_mesh(mesh, scale)
```

Arguments

mesh	A mesh descriptor list.
scale	A single numeric scale factor (uniform) or a length-3 numeric vector for per-axis scaling (x, y, z).

Value

A new mesh descriptor list with scaled vertices.

Examples

```
mesh <- generate_cuboid(c(0, 0, 0), c(1, 1, 1))
big <- scale_mesh(mesh, 3)
flat <- scale_mesh(mesh, c(2, 0.5, 1))
```

scene *Create a scene descriptor*

Description

Bundles a list of meshes together with their placement transforms, a camera, and render options into a single scene object that can be passed to render_scene() or write_gltf().

Usage

```
scene(meshes, camera = NULL, options = NULL, transforms = NULL, names = NULL)
```

Arguments

meshes	A list of mesh descriptors (scimesh or rgl format, see render_scene()), or a list of already-built scene nodes (list(mesh = ..., transform = ..., name = ...)).
camera	A camera list from camera() or camera_auto(). Optional here; when NULL it must be supplied when calling render_scene().
options	A render options list from render_options(). Optional here; when NULL, render_scene() uses its default options.
transforms	Optional list of 4x4 numeric matrices, one per mesh, overriding any embedded transform in the nodes. May contain NULL entries to mean identity.
names	Optional character vector, one per mesh, overriding any embedded name.

Details

Each mesh is wrapped into a scene node list(mesh = ..., transform = ...,name = ...). The optional transform is a 4x4 numeric matrix (column-major, GLM style — the same convention accepted by transform_mesh()) that places the mesh in world space at render/export time without modifying the mesh itself. The optional name is used by exporters such as glTF for node names.

Value

A scene descriptor list with S3 class "scimesh_scene", with components meshes (list of scene nodes), camera, and options.

Examples

```
cube1 <- generate_cuboid(c(0, 0, 0), c(0.5, 0.5, 0.5), c(1, 0, 0, 1))
cube2 <- generate_cuboid(c(0, 0, 0), c(0.5, 0.5, 0.5), c(0, 0, 1, 1))
# place the second cube 2 units along +X
tr <- diag(1, 4); tr[1, 4] <- 2
sc <- scene(list(cube1, list(mesh = cube2, transform = tr, name = "blue")),
            camera = camera_auto(list(cube1, cube2), direction = c(1, 1, 1)))
img <- render_scene(sc)
```

---

stack_horizontal *Stack images horizontally*

Description

Stacks a list of rendered images side by side. A convenience wrapper around compose_layout().

Usage

```
stack_horizontal(
  ...,
  colorbar = NULL,
  colorbar_height = 80L,
  colorbar_width = 80L,
  background = c(0, 0, 0, 0),
  colorbar_side = c("right", "left"),
  crop = FALSE
)
```

Arguments

`...`	Images from `render_mesh()` or `render_scene()`, or a list of images.
`colorbar`	Optional colorbar.
`colorbar_height`	Height of the colorbar in pixels.
`colorbar_width`	Width of the colorbar in pixels.
`background`	Background RGBA color for padding.
`colorbar_side`	Side for the colorbar: `"right"` (default) or `"left"`.
`crop`	If `TRUE`, crop whitespace from the output.

Value

A composed image list.

Examples

```
mesh1 <- generate_cuboid(c(-2, 0, 0), c(0.5, 1, 1), c(1, 0, 0, 1))
mesh2 <- generate_cuboid(c( 2, 0, 0), c(0.5, 1, 1), c(0, 0, 1, 1))
img1 <- render_mesh(mesh1$vertices, mesh1$triangles)
img2 <- render_mesh(mesh2$vertices, mesh2$triangles)
result <- stack_horizontal(img1, img2)
tmp_file <- tempfile(fileext = ".png")
write_png(result, tmp_file)
```

`stack_vertical` *Stack images vertically*

Description

Stacks a list of rendered images vertically (one below another). A convenience wrapper around `compose_layout()`.

Usage

```
stack_vertical(
  ...,
  colorbar = NULL,
  colorbar_height = 80L,
  colorbar_width = 80L,
  background = c(0, 0, 0, 0),
  colorbar_side = c("right", "left"),
  crop = FALSE
)
```

Arguments

...	Images from `render_mesh()` or `render_scene()`, or a list of images.
colorbar	Optional colorbar from `colorbar_horizontal()` or `colorbar_vertical()`.
colorbar_height	Height of the colorbar in pixels.
colorbar_width	Width of the colorbar in pixels.
background	Background RGBA color for padding.
colorbar_side	Side for the colorbar: `"right"` (default) or `"left"`.
crop	If `TRUE`, crop whitespace from the output.

Value

A composed image list.

Examples

```
mesh1 <- generate_cuboid(c(0, 2, 0), c(1, 0.5, 1), c(1, 0, 0, 1))
mesh2 <- generate_cuboid(c(0, -2, 0), c(1, 0.5, 1), c(0, 1, 0, 1))
img1 <- render_mesh(mesh1$vertices, mesh1$triangles)
img2 <- render_mesh(mesh2$vertices, mesh2$triangles)
result <- stack_vertical(img1, img2)
tmp_file <- tempfile(fileext = ".png")
write_png(result, tmp_file)
```

---

transform_mesh *Apply a 4x4 transformation matrix to a mesh*

Description

Transforms all vertex positions in a mesh by a 4x4 homogeneous matrix (applied as `M * (x, y, z, 1)^T`). Vertex colors and normals are untouched.

Usage

```
transform_mesh(mesh, matrix)
```

Arguments

mesh	A mesh descriptor list with vertices and triangles, as returned by render_mesh() or built by scimesh_generate_multi_spheres() etc.
matrix	A 4x4 numeric matrix.

Value

A new mesh descriptor list with transformed vertices.

Examples

```
mesh <- generate_cuboid(c(0, 0, 0), c(1, 1, 1))
mat <- diag(4)
mat[1:3, 4] <- c(2, 3, 4)
translated <- transform_mesh(mesh, mat)
translated$vertices[1, ]
```

---

translate_mesh *Translate a mesh*

Description

Translate a mesh

Usage

```
translate_mesh(mesh, translation)
```

Arguments

mesh	A mesh descriptor list.
translation	Length-3 numeric vector (x, y, z).

Value

A new mesh descriptor list with translated vertices.

Examples

```
mesh <- generate_cuboid(c(0, 0, 0), c(1, 1, 1))
moved <- translate_mesh(mesh, c(5, 0, 0))
colMeans(moved$vertices)
```

viridis_colormap *Viridis colormap*

Description

Returns the viridis color palette. A convenience wrapper around `grDevices::hcl.colors` that mimics the viridis color scheme without requiring extra packages.

Usage

```
viridis_colormap(n, alpha = 1, direction = 1)
```

Arguments

`n`	Number of colors.
`alpha`	Alpha channel value (0-1).
`direction`	Forward (1) or reversed (-1) direction.

Value

A character vector of hex color strings.

Examples

```
cols <- viridis_colormap(10)
plot(1:10, pch = 19, col = cols, cex = 3)
```

write_gltf *Write a scene or mesh list to a glTF file*

Description

Exports a scene (or a list of meshes) to the glTF 2.0 format, either as a JSON document with an external binary buffer (`.gltf` + `.bin`) or as a single self-contained binary file (`.glb`). The resulting file can be viewed in any glTF-capable viewer (e.g. a browser using three.js) and includes per-mesh placement transforms, vertex colors, and optionally a camera.

Usage

```
write_gltf(meshes, path, camera = NULL, format = c("gltf", "glb"))
```

Arguments

meshes	Either a scimesh_scene object (see scene()), a list of mesh descriptors, or a list of scene nodes — the same inputs accepted by render_scene().
path	Output file path. For format = "gltf" the binary buffer is written next to it as <stem>.bin.
camera	Optional camera list from camera() or camera_auto(). When provided, a glTF perspective camera node is included.
format	Output format: "gltf" (default, JSON + .bin) or "glb" (single binary file).

Details

Renderer-specific settings (shading mode, fog, SSAO, ...) are not part of the glTF standard and are not exported. Per-face colors are exported by splitting vertices (each triangle gets its own vertices), which increases geometry roughly 3x.

Value

Invisibly NULL.

Examples

```
cube <- generate_cuboid(c(0, 0, 0), c(0.5, 0.5, 0.5), c(1, 0, 0, 1))
tr <- diag(1, 4); tr[1, 4] <- 2
sc <- scene(list(cube, list(mesh = cube, transform = tr, name = "second")))
out <- tempfile(fileext = ".glb")
write_gltf(sc, out, format = "glb")
```

---

write_png *Write a rendered image to a PNG file*

Description

Writes the output of render_mesh() or render_scene() to a PNG file using the built-in C++ PNG writer (stb_image_write). No additional R packages are required.

Usage

```
write_png(image, filename)
```

Arguments

image	An image list returned by render_mesh() or render_scene().
filename	Output PNG file path.

Value

No return value; called for side effects.

Examples

```
mesh <- generate_cuboid(c(0, 0, 0), c(1, 1, 1))
img <- render_mesh(mesh$vertices, mesh$triangles)
tmp_file <- tempfile(fileext = ".png")
write_png(img, tmp_file)
```

`write_stl` *Write a mesh to an STL file*

Description

Writes a scimesh mesh descriptor to an ASCII or binary STL file.

Usage

```
write_stl(mesh, path, format = c("binary", "ascii"))
```

Arguments

`mesh`	A mesh descriptor list.
`path`	Path to the output STL file.
`format`	`"binary"` (default) or `"ascii"`.

Value

invisible NULL, called for side effects of writing the file.

Examples

```
mesh <- generate_cuboid(c(0, 0, 0), c(1, 1, 1))
tmp_file <- tempfile(fileext = ".stl")
write_stl(mesh, tmp_file, format = "binary")
```

write_tga *Write a rendered image to a TGA file*

Description

Writes the output of `render_mesh()` or `render_scene()` to a TGA file using scimesh's own C++ TGA writer (no external dependencies). TGA output is uncompressed true-color.

Usage

```
write_tga(image, filename, use24bit = FALSE)
```

Arguments

`image`	An image list returned by `render_mesh()` or `render_scene()`.
`filename`	Output TGA file path.
`use24bit`	If `TRUE`, write 24-bit RGB (no alpha channel). The default `FALSE` writes 32-bit RGBA.

Value

No return value; called for side effects.

Examples

```
mesh <- generate_cuboid(c(0, 0, 0), c(1, 1, 1))
img <- render_mesh(mesh$vertices, mesh$triangles)
tmp_file <- tempfile(fileext = ".tga")
write_tga(img, tmp_file)
```

Index