Package 'rayvertex'

February 3, 2025

Type Package

Title 3D Software Rasterizer

Version 0.12.0

Date 2025-01-02

Maintainer Tyler Morgan-Wall <tylermw@gmail.com>

Description Rasterize images using a 3D software renderer. 3D scenes are created either by importing external files, building scenes out of the included objects, or by constructing meshes manually. Supports point and directional lights, anti-aliased lines, shadow mapping, transparent objects, translucent objects, multiple materials types, reflection, refraction, environment maps, multicore rendering, bloom, tone-mapping, and screen-space ambient occlusion.

License GPL (>= 3)

Copyright file inst/COPYRIGHTS

Depends R (>= 4.1)

Imports Rcpp (>= 1.0.6), grDevices, rayimage (>= 0.15.1), png, digest, pillar (>= 1.10.1), vctrs, tibble, withr, cli

Suggests Rvcg, magick, raster, testthat (>= 3.0.0)

LinkingTo Rcpp, spacefillr, RcppThread, rayimage

RoxygenNote 7.3.2

URL https://www.rayvertex.com, https://github.com/tylermorganwall/rayvertex

BugReports https://github.com/tylermorganwall/rayvertex/issues

Encoding UTF-8

SystemRequirements C++17

Config/testthat/edition 3

NeedsCompilation yes

Author Tyler Morgan-Wall [aut, cph, cre] (<https://orcid.org/0000-0002-3131-3814>), Syoyo Fujita [ctb, cph], Vilya Harvey [ctb, cph], G-Truc Creation [ctb, cph], Sean Barrett [ctb, cph]

Repository CRAN

Date/Publication 2025-02-03 08:20:02 UTC

Contents

`add_light` *Add light*

Description

Add light

Usage

```
add_light(lights, light)
```

Arguments

`lights`	Current light scene.
`light`	New light to add.

Value

A matrix representing the light information.

Examples

```
if(run_documentation()) {
#Add a light to scene (manually specify the light automatically added to the Cornell Box
lights = point_light(position=c(555/2,450,555/2),
                     falloff_quad = 0.0, constant = 0.0002, falloff = 0.005)
generate_cornell_mesh(light=FALSE) |>
 rasterize_scene(light_info = lights)

#Add directional lights and a point light
lights_d = add_light(lights, directional_light(direction=c(1,1.5,-1), intensity=0.2)) |>
 add_light(directional_light(direction=c(-1,1.5,-1),color="red", intensity=0.2)) |>
 add_light(point_light(position=c(555/2,50,555/2), color="blue", intensity=0.3,
                       falloff_quad = 0.0, constant = 0.0002, falloff = 0.005))

generate_cornell_mesh(light=FALSE) |>
 rasterize_scene(light_info = lights_d)
}
```

add_lines *Add Line*

Description

Add Line

Usage

```
add_lines(lines, line)
```

Arguments

lines	Existing lines or empty (0-row) matrix.
line	Line to add, generated with `generate_line()`

Value

New line matrix.

Examples

```
if(run_documentation()) {
#Generate a cube out of lines
cube_outline = generate_line(start = c(-1, -1, -1), end = c(-1, -1, 1)) |>
  add_lines(generate_line(start = c(-1, -1, -1), end = c(-1, 1, -1))) |>
  add_lines(generate_line(start = c(-1, -1, -1), end = c(1, -1, -1))) |>
  add_lines(generate_line(start = c(-1, -1, 1), end = c(-1, 1, 1))) |>
  add_lines(generate_line(start = c(-1, -1, 1), end = c(1, -1, 1))) |>
  add_lines(generate_line(start = c(-1, 1, 1), end = c(-1, 1, -1))) |>
  add_lines(generate_line(start = c(-1, 1, 1), end = c(1, 1, 1))) |>
  add_lines(generate_line(start = c(1, 1, -1), end = c(1, -1, -1))) |>
  add_lines(generate_line(start = c(1, 1, -1), end = c(1, 1, 1))) |>
  add_lines(generate_line(start = c(1, -1, -1), end = c(1, -1, 1))) |>
  add_lines(generate_line(start = c(1, -1, 1), end = c(1, 1, 1))) |>
  add_lines(generate_line(start = c(-1, 1, -1), end = c(1, 1, -1)))

rasterize_lines(cube_outline,fov=90,lookfrom=c(0,0,3))
}
```

add_plane_uv_mesh *Add Plane UV Mapping to Mesh*

Description

Applies a planar UV mapping to a mesh based on a given direction and set of U/V vectors. If full_mesh_bbox is true, the UV mapping is scaled based on the bounding box of the entire mesh. If false, each shape's bounding box is used. One of direction/u/v must be NULL and will be calculated from the others.

Usage

```
add_plane_uv_mesh(
  mesh,
  direction = c(0, 1, 0),
  u = NULL,
  v = NULL,
  override_existing = FALSE,
  full_mesh_bbox = TRUE
)
```

Arguments

mesh	The mesh to which the UV mapping will be applied.
direction	Default c(0, 1, 0). A vector specifying the direction for UV mapping. If not specified and u/v are both specified, this will be ignored.
u	Default NULL. A vector specifying the u direction.
v	Default NULL. A vector specifying the v direction.
override_existing	Default FALSE. Specifies whether existing UV coordinates should be overridden.
full_mesh_bbox	Default TRUE. Specifies whether the full mesh's bounding box is used for UV mapping.

Value

Modified mesh with added UV mapping.

Examples

```
if(run_documentation()) {
#Let's construct a mesh from the volcano dataset
#Build the vertex matrix
vertex_list = list()
counter = 1
for(i in 1:nrow(volcano)) {
  for(j in 1:ncol(volcano)) {
```

```
    vertex_list[[counter]] = matrix(c(j,volcano[i,j]/3,i), ncol=3)
    counter = counter + 1
  }
}
vertices = do.call(rbind,vertex_list)

#Build the index matrix
index_list = list()
counter = 0
for(i in 1:(nrow(volcano)-1)) {
  for(j in 1:(ncol(volcano)-1)) {
    index_list[[counter+1]] = matrix(c(counter,counter+ncol(volcano),counter+1,
                              counter+ncol(volcano),counter+ncol(volcano)+1,counter + 1),
                                     nrow=2, ncol=3, byrow=TRUE)
    counter = counter + 1
  }
  counter = counter + 1
}
indices = do.call("rbind",index_list)

#Create a checkerboard image
create_checkerboard_texture = function(filename, n = 16) {
  old_par = par(no.readonly = TRUE)
  on.exit(par(old_par))
  plot.new()
  par(mar = c(0, 0, 0, 0))
  checkerboard = matrix(c(1, 0), nrow = n+1, ncol = n)
  png(filename, width = 800, height = 800)
  image(1:(n+1), 1:n, checkerboard, col = c("dodgerblue", "red"),
        axes = FALSE, xlab = "", ylab = "")
  dev.off()
}
checkerboard_file = tempfile(fileext = ".png")
create_checkerboard_texture(checkerboard_file)
rayimage::plot_image(checkerboard_file)
}

if(run_documentation()) {
#Construct the mesh
volc_mesh = construct_mesh(vertices = vertices, indices = indices,
                           material = material_list(type="phong", diffuse="darkred",
                                                    ambient = "darkred", ambient_intensity=0.2))


#Set the direction so that the checkerboard will be mapped to the surface like a carpet
uv = add_plane_uv_mesh(volc_mesh, direction=c(0,200,0), u = c(1,0,0))
uv = set_material(uv, texture_location = checkerboard_file,
                  ambient = "white", ambient_intensity=0.1)
#Rasterize the scene
rasterize_scene(center_mesh(uv), lookfrom=c(200,200,200),fov=0,width=1200,height=1200,
                light_info = directional_light(c(0,1,1)) |>
                  add_light(directional_light(c(1,1,-1))),ortho_dimensions=c(120,120))
}
```

```
if(run_documentation()) {
#Set the direction so that the checkerboard will be mapped directly at the camera
uv = add_plane_uv_mesh(volc_mesh, direction=c(200,200,200), v = c(-1,1,-1))
uv = set_material(uv, texture_location = checkerboard_file,
                  ambient = "white", ambient_intensity=0.1)
#Rasterize the scene
rasterize_scene(center_mesh(uv), lookfrom=c(200,200,200),fov=0,width=1200,height=1200,
                light_info = directional_light(c(0,1,1)) |>
                add_light(directional_light(c(1,1,-1))), ortho_dimensions=c(120,120))
}
```

---

add_shape *Add Shape*

Description

Add shape to the scene.

Usage

```
add_shape(scene, shape = NULL)
```

Arguments

scene	The scene to add the shape.
shape	The mesh to add to the scene.

Value

Scene with shape added.

Examples

```
if(run_documentation()) {
#Generate several spheres in the cornell box
scene = generate_cornell_mesh()
set.seed(1)

for(i in 1:30) {
 col = hsv(runif(1))
 scene = add_shape(scene, sphere_mesh(position=runif(3)*400+155/2,
                                      material=material_list(diffuse=col, type="phong",
                                                             ambient=col,ambient_intensity=0.2),
                                      radius=30))
}
rasterize_scene(scene, light_info=directional_light(direction=c(0.1,0.6,-1)))
}
```

add_sphere_uv_mesh *Add Sphere UV Mapping to Mesh*

Description

Applies a planar UV mapping to a mesh based on a spherical direction from the origin.

Usage

```
add_sphere_uv_mesh(mesh, origin = c(0, 0, 0), override_existing = FALSE)
```

Arguments

mesh	The mesh to which the UV mapping will be applied.
origin	Default c(0, 0, 0). A vector specifying the origin to apply spherical UV coordinates.
override_existing	Default FALSE. Specifies whether existing UV coordinates should be overridden.

Value

Modified mesh with added UV mapping.

Examples

```
if(run_documentation()) {
#Let's construct a mesh from the volcano dataset

}
```

arrow_mesh *Arrow 3D Model*

Description

Arrow 3D Model

Usage

```
arrow_mesh(
  start = c(0, 0, 0),
  end = c(0, 1, 0),
  radius_top = 0.5,
  radius_tail = 0.25,
  tail_proportion = 0.5,
  direction = NA,
```

```
  from_center = TRUE,
  material = material_list()
)
```

Arguments

start	Default `c(0, 0, 0)`. Base of the arrow, specifying x, y, z.
end	Default `c(0, 1, 0)`. Tip of the arrow, specifying x, y, z.
radius_top	Default `0.5`. Radius of the top of the arrow.
radius_tail	Default `0.2`. Radius of the tail of the arrow.
tail_proportion	Default `0.5`. Proportion of the arrow that is the tail.
direction	Default `NA`. Alternative to `start` and end, specify the direction (via a length-3 vector) of the arrow. Arrow will be centered at `start`, and the length will be determined by the magnitude of the direction vector.
from_center	Default `TRUE`. If orientation specified via `direction`, setting this argument to `FALSE` will make `start` specify the bottom of the cone, instead of the middle.
material	Default `material_list()` (default values). Specify the material of the object.

Value

List describing the mesh.

Examples

```
if(run_documentation()) {
#Generate an arrow
generate_cornell_mesh() |>
 add_shape(arrow_mesh(start = c(555/2, 20, 555/2), end = c(555/2, 300, 555/2), radius_tail=50,
                      radius_top = 100,
                     material = material_list(diffuse="dodgerblue"))) |>
  rasterize_scene(light_info = directional_light(c(0.5,0.5,-1)))
}
if(run_documentation()) {
#Generate a blue arrow with a wide tail
generate_cornell_mesh() |>
 add_shape(arrow_mesh(start = c(555/2, 20, 555/2), end = c(555/2, 300, 555/2), radius_tail=100,
                      radius_top = 150,
                     material = material_list(diffuse="dodgerblue"))) |>
  rasterize_scene(light_info = directional_light(c(0.5,0.5,-1)))
  }
if(run_documentation()) {
#Generate a long, thin arrow and change the proportions
generate_cornell_mesh() |>
 add_shape(arrow_mesh(start = c(555/2, 20, 555/2), end = c(555/2, 400, 555/2), radius_top=30,
                      radius_tail = 10, tail_proportion = 0.8,
                     material = material_list(diffuse="dodgerblue"))) |>
  rasterize_scene(light_info = directional_light(c(0.5,0.5,-1)))
}
```

```
if(run_documentation()) {
#Change the start and end points
generate_cornell_mesh() |>
  add_shape(arrow_mesh(start = c(500, 20, 555/2), end = c(50, 500, 555/2), radius_top=30,
                       radius_tail = 10, tail_proportion = 0.8,
                      material = material_list(diffuse="dodgerblue"))) |>
  add_shape(arrow_mesh(start = c(500, 500, 500), end = c(50, 50, 50), radius_top=30,
                       radius_tail = 10, tail_proportion = 0.8,
                      material = material_list(diffuse="red"))) |>
  add_shape(arrow_mesh(start = c(555/2, 50, 500), end = c(555/2, 50, 50), radius_top=30,
                       radius_tail = 10, tail_proportion = 0.8,
                      material = material_list(diffuse="green"))) |>
  rasterize_scene(light_info = directional_light(c(0.5,0.5,-1)))
}
```

`center_mesh` *Center Mesh*

Description

Centers the mesh at the origin.

Usage

```
center_mesh(mesh)
```

Arguments

`mesh`	The mesh object.

Value

Centered mesh

Examples

```
if(run_documentation()) {
#Center the Cornell box and the R OBJ at the origin
center_mesh(generate_cornell_mesh()) |>
  add_shape(center_mesh(obj_mesh(r_obj(),scale=100,angle=c(0,180,0)))) |>
  rasterize_scene(lookfrom=c(0,0,-1100),fov=40,lookat=c(0,0,0),
                  light_info = directional_light(c(0.4,0.4,-1)) |>
    add_light(point_light(c(0,450,0),  falloff_quad = 0.0, constant = 0.0002, falloff = 0.005)))
}
```

change_material *Change Material*

Description

Change individual material properties, leaving others alone.

Usage

```
change_material(
  mesh,
  id = NULL,
  sub_id = 1,
  diffuse = NULL,
  ambient = NULL,
  specular = NULL,
  transmittance = NULL,
  emission = NULL,
  shininess = NULL,
  ior = NULL,
  dissolve = NULL,
  illum = NULL,
  texture_location = NULL,
  normal_texture_location = NULL,
  bump_texture_location = NULL,
  specular_texture_location = NULL,
  ambient_texture_location = NULL,
  emissive_texture_location = NULL,
  diffuse_intensity = NULL,
  bump_intensity = NULL,
  specular_intensity = NULL,
  emission_intensity = NULL,
  ambient_intensity = NULL,
  culling = NULL,
  type = NULL,
  translucent = NULL,
  toon_levels = NULL,
  toon_outline_width = NULL,
  toon_outline_color = NULL,
  reflection_intensity = NULL,
  reflection_sharpness = NULL,
  two_sided = NULL
)
```

Arguments

mesh	Mesh to change.

`id` Default `NULL`. Either a number specifying the material to change, or a character vector matching the material name.

`sub_id` Default `1`. A number specifying which material to change (within an id).

`diffuse` Default `NULL`. The diffuse color.

`ambient` Default `NULL`. The ambient color.

`specular` Default `NULL`. The specular color.

`transmittance` Default `NULL`. The transmittance

`emission` Default `NULL`. The emissive color.

`shininess` Default `NULL`. The shininess exponent.

`ior` Default `NULL`. The index of refraction. If this is not equal to `1.0`, the material will be refractive.

`dissolve` Default `NULL`. The transparency.

`illum` Default `NULL`. The illumination.

`texture_location`
Default `NULL`. The diffuse texture location.

`normal_texture_location`
Default `NULL`. The normal texture location.

`bump_texture_location`
Default `NULL`. The bump texture location.

`specular_texture_location`
Default `NULL`. The specular texture location.

`ambient_texture_location`
Default `NULL`. The ambient texture location.

`emissive_texture_location`
Default `NULL`. The emissive texture location.

`diffuse_intensity`
Default `NULL`. The diffuse intensity.

`bump_intensity` Default `NULL`. The bump intensity.

`specular_intensity`
Default `NULL`. The specular intensity.

`emission_intensity`
Default `NULL`. The emission intensity.

`ambient_intensity`
Default `NULL`. The ambient intensity.

`culling` Default `NULL`. The culling type. Options are `back`, `front`, and `none`.

`type` Default `NULL`. The shader type. Options include `diffuse`,`phong`,`vertex`, and `color`.

`translucent` Default `NULL`. Whether light should transmit through a semi-transparent material.

`toon_levels` Default `NULL`. Number of color breaks in the toon shader.

`toon_outline_width`
Default `NULL`. Expansion term for model to specify toon outline width. Note: setting this property via this function currently does not generate outlines. Specify it during object creation.

toon_outline_color
: Default NULL. Toon outline color.Note: setting this property via this function currently does not color outlines. Specify it during object creation.

reflection_intensity
: Default NULL. Intensity of the reflection of the environment map, if present. This will be ignored if the material is refractive.

reflection_sharpness
: Default NULL. Sharpness of the reflection, where lower values have blurrier reflections. Must be greater than zero and less than one.

two_sided
: Default NULL. Whether diffuse materials should be two sided (normal is taken as the absolute value of the dot product of the light direction and the normal).

Value

Shape with new material settings

Examples

```
if(run_documentation()) {
p_sphere = sphere_mesh(position=c(555/2,555/2,555/2),
                       radius=40,material=material_list(diffuse="purple"))
generate_cornell_mesh() |>
 add_shape(translate_mesh(p_sphere,c(0,-100,0))) |>
 add_shape(change_material(translate_mesh(p_sphere,c(200,-100,0)),diffuse="red")) |>
 add_shape(change_material(translate_mesh(p_sphere,c(100,-100,0)),dissolve=0.5)) |>
 add_shape(change_material(translate_mesh(p_sphere,c(-100,-100,0)),type="phong")) |>
 add_shape(change_material(translate_mesh(p_sphere,c(-200,-100,0)),type="phong",shininess=30)) |>
 rasterize_scene(light_info=directional_light(direction=c(0.1,0.6,-1)))
}

if(run_documentation()) {
#Change several shapes at once
p_sphere |>
 add_shape(change_material(translate_mesh(p_sphere,c(200,0,0)),diffuse="red")) |>
 add_shape(change_material(translate_mesh(p_sphere,c(100,0,0)),dissolve=0.5)) |>
 add_shape(change_material(translate_mesh(p_sphere,c(-100,0,0)),type="phong")) |>
 add_shape(change_material(translate_mesh(p_sphere,c(-200,0,0)),type="phong",shininess=30)) |>
 change_material(diffuse = "red") |>
 add_shape(generate_cornell_mesh()) |>
 rasterize_scene(light_info=directional_light(direction=c(0.1,0.6,-1)))
}
```

---

color_lines *Color Lines*

Description

Color Lines

Usage

```
color_lines(lines, color = "white")
```

Arguments

lines	The line scene.
color	Default white. The color to convert the lines to.

Value

Colored line matrix.

Examples

```
if(run_documentation()) {
#Generate a cube out of lines
cube_outline = generate_line(start = c(-1, -1, -1), end = c(-1, -1, 1)) |>
  add_lines(generate_line(start = c(-1, -1, -1), end = c(-1, 1, -1))) |>
  add_lines(generate_line(start = c(-1, -1, -1), end = c(1, -1, -1))) |>
  add_lines(generate_line(start = c(-1, -1, 1), end = c(-1, 1, 1))) |>
  add_lines(generate_line(start = c(-1, -1, 1), end = c(1, -1, 1))) |>
  add_lines(generate_line(start = c(-1, 1, 1), end = c(-1, 1, -1))) |>
  add_lines(generate_line(start = c(-1, 1, 1), end = c(1, 1, 1))) |>
  add_lines(generate_line(start = c(1, 1, -1), end = c(1, -1, -1))) |>
  add_lines(generate_line(start = c(1, 1, -1), end = c(1, 1, 1))) |>
  add_lines(generate_line(start = c(1, -1, -1), end = c(1, -1, 1))) |>
  add_lines(generate_line(start = c(1, -1, 1), end = c(1, 1, 1))) |>
  add_lines(generate_line(start = c(-1, 1, -1), end = c(1, 1, -1)))

cube_outline |>
  color_lines(color="red") |>
  rasterize_lines()
}
```

---

cone_mesh *Cone 3D Model*

Description

Cone 3D Model

Usage

```
cone_mesh(
  start = c(0, 0, 0),
  end = c(0, 1, 0),
  radius = 0.5,
  direction = NA,
```

```
  from_center = FALSE,
  material = material_list()
)
```

Arguments

start	Default c(0, 0, 0). Base of the cone, specifying x, y, z.
end	Default c(0, 1, 0). Tip of the cone, specifying x, y, z.
radius	Default 1. Radius of the bottom of the cone.
direction	Default NA. Alternative to start and end, specify the direction (via a length-3 vector) of the cone. Cone will be centered at start, and the length will be determined by the magnitude of the direction vector.
from_center	Default TRUE. If orientation specified via direction, setting this argument to FALSE will make start specify the bottom of the cone, instead of the middle.
material	Default material_list() (default values). Specify the material of the object.

Value

List describing the mesh.

Examples

```
if(run_documentation()) {
#Generate a cone
generate_cornell_mesh() |>
  add_shape(cone_mesh(start = c(555/2, 20, 555/2), end = c(555/2, 300, 555/2),
                      radius = 100)) |>
  rasterize_scene(light_info = directional_light(c(0.5,0.5,-1)))
}
if(run_documentation()) {
#Generate a blue cone with a wide base
generate_cornell_mesh() |>
  add_shape(cone_mesh(start = c(555/2, 20, 555/2), end = c(555/2, 300, 555/2), radius=200,
                      material = material_list(diffuse="dodgerblue"))) |>
  rasterize_scene(light_info = directional_light(c(0.5,0.5,-1)))
}
if(run_documentation()) {
#Generate a long, thin cone
generate_cornell_mesh() |>
  add_shape(cone_mesh(start = c(555/2, 20, 555/2), end = c(555/2, 400, 555/2), radius=50,
                      material = material_list(diffuse="dodgerblue"))) |>
  rasterize_scene(light_info = directional_light(c(0.5,0.5,-1)))
}
```

construct_mesh *Manually construct a mesh*

Description

Manually construct a mesh

Usage

```
construct_mesh(
  vertices,
  indices,
  normals = NULL,
  norm_indices = NULL,
  texcoords = NULL,
  tex_indices = NULL,
  material = material_list()
)
```

Arguments

vertices	Nx3 matrix of vertex coordinates..
indices	Nx3 integer matrix, where each row defines a triangle using the vertices defined in vertices.
normals	Default NULL. Nx3 matrix of normals.
norm_indices	Nx3 integer matrix, where each row defines the normal for a vertex using the normals defined in normals for the corresponding triangle in indices. Required to be the same number of rows as indices.
texcoords	Default NULL. Nx2 matrix of texture coordinates.
tex_indices	Nx3 integer matrix, where each row defines the texture coordinates for a triangle using the tex coords defined in texcoors for the corresponding triangle in indices. Required to be the same number of rows as indices.
material	Default material_list() (default values). Specify the material of the object.

Value

List containing mesh info.

Examples

```
if(run_documentation()) {
#Let's construct a mesh from the volcano dataset
#Build the vertex matrix
vertex_list = list()
counter = 1
for(i in 1:nrow(volcano)) {
```

```
  for(j in 1:ncol(volcano)) {
    vertex_list[[counter]] = matrix(c(j,volcano[i,j],i), ncol=3)
    counter = counter + 1
  }
}
vertices = do.call(rbind,vertex_list)

#Build the index matrix
index_list = list()
counter = 0
for(i in 1:(nrow(volcano)-1)) {
  for(j in 1:(ncol(volcano)-1)) {
    index_list[[counter+1]] = matrix(c(counter,counter+ncol(volcano),counter+1,
                              counter+ncol(volcano),counter+ncol(volcano)+1,counter + 1),
                                     nrow=2, ncol=3, byrow=TRUE)
    counter = counter + 1
  }
  counter = counter + 1
}
indices = do.call(rbind,index_list)

#Construct the mesh
volc_mesh = construct_mesh(vertices = vertices, indices = indices,
                           material = material_list(type="phong", diffuse="darkred",
                                                    ambient = "darkred", ambient_intensity=0.2))

#Rasterize the scene
rasterize_scene(volc_mesh, lookfrom=c(-50,230,100),fov=60,width=1200,height=1200,
                light_info = directional_light(c(0,1,1)) |>
                  add_light(directional_light(c(1,1,-1))))
}
```

---

cube_mesh *Cube 3D Model*

Description

3D obj model of the letter R

Usage

```
cube_mesh(
  position = c(0, 0, 0),
  scale = c(1, 1, 1),
  angle = c(0, 0, 0),
  pivot_point = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  material = material_list()
)
```

Arguments

position	Default c(0,0,0). Position of the mesh.
scale	Default c(1,1,1). Scale of the mesh. Can also be a single numeric value scaling all axes uniformly.
angle	Default c(0,0,0). Angle to rotate the mesh.
pivot_point	Default c(0,0,0). Point around which to rotate the mesh.
order_rotation	Default c(1,2,3). Order to rotate the axes.
material	Default material_list() (default values). Specify the material of the object.

Value

List describing the mesh.

Examples

```
if(run_documentation()) {
#Generate a cube
generate_cornell_mesh() |>
  add_shape(cube_mesh(position = c(555/2, 100, 555/2), scale = 100)) |>
  rasterize_scene(light_info = directional_light(c(0.5,0.5,-1)))
}
if(run_documentation()) {
#Generate a blue rotated cube
generate_cornell_mesh() |>
  add_shape(cube_mesh(position = c(555/2, 100, 555/2), scale = 100, angle=c(0,45,0),
                      material = material_list(diffuse="dodgerblue"))) |>
  rasterize_scene(light_info = directional_light(c(0.5,0.5,-1)))
}
if(run_documentation()) {
#Generate a scaled, blue rotated cube
generate_cornell_mesh() |>
  add_shape(cube_mesh(position = c(555/2, 100, 555/2), angle=c(0,45,0),
                      scale = c(2,0.5,0.8)*100,
                      material = material_list(diffuse="dodgerblue"))) |>
  rasterize_scene(light_info = directional_light(c(0.5,0.5,-1)))
}
```

---

cylinder_mesh *Cylinder 3D Model*

Description

Cylinder 3D Model

Usage

```
cylinder_mesh(
  position = c(0, 0, 0),
  radius = 0.5,
  length = 1,
  angle = c(0, 0, 0),
  pivot_point = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  material = material_list()
)
```

Arguments

position	Default `c(0,0,0)`. Position of the mesh.
radius	Default `0.5`. Radius of the cylinder.
length	Default `1`. Length of the cylinder.
angle	Default `c(0,0,0)`. Angle to rotate the mesh.
pivot_point	Default `c(0,0,0)`. Point around which to rotate the mesh.
order_rotation	Default `c(1,2,3)`. Order to rotate the axes.
material	Default `material_list()` (default values). Specify the material of the object.

Value

List describing the mesh.

Examples

```
if(run_documentation()) {
#Generate a cylinder
generate_cornell_mesh() |>
  add_shape(cylinder_mesh(position=c(555/2,150,555/2),
                 radius = 50, length=300, material = material_list(diffuse="purple"))) |>
  rasterize_scene(light_info = directional_light(c(0.5,0.5,-1)))
}
if(run_documentation()) {
#Generate a wide, thin disk
generate_cornell_mesh() |>
  add_shape(cylinder_mesh(position=c(555/2,20,555/2),
                  radius = 200, length=5, material = material_list(diffuse="purple"))) |>
  rasterize_scene(light_info = directional_light(c(0.5,0.5,-1)))
}
if(run_documentation()) {
#Generate a narrow cylinder
generate_cornell_mesh() |>
  add_shape(cylinder_mesh(position=c(555/2,555/2,555/2),angle=c(45,-45,0),
                 radius = 10, length=500, material = material_list(diffuse="purple"))) |>
  rasterize_scene(light_info = directional_light(c(0.5,0.5,-1)))
}
```

`directional_light` *Generate Directional Lights*

Description

Generate Directional Lights

Usage

```
directional_light(direction = c(0, 1, 0), color = "white", intensity = 1)
```

Arguments

`direction`	Default `c(0,1,0)`. Direction of the light.
`color`	Default `white`. COlor of the light.
`intensity`	Default `1`. Intensity of the light.

Value

A matrix representing the light information.

Examples

```
if(run_documentation()) {
#Add a light to scene (manually specify the light automatically added to the Cornell Box
lights = point_light(position=c(555/2,450,555/2),
                     falloff_quad = 0.0, constant = 0.0002, falloff = 0.005)
generate_cornell_mesh(light=FALSE) |>
 rasterize_scene(light_info = lights)

#Add a directional light
lights_d = add_light(lights, directional_light(direction=c(1,1.5,-1)))

generate_cornell_mesh(light=FALSE) |>
 rasterize_scene(light_info = lights_d)

#Change the intensity and color
lights_d = add_light(lights,
                 directional_light(direction=c(1,1.5,-1),color="orange", intensity=0.5))

generate_cornell_mesh(light=FALSE) |>
 rasterize_scene(light_info = lights_d)
}
```

displacement_sphere *Construct Displacement Sphere*

Description

Construct Displacement Sphere

Usage

```
displacement_sphere(
  displacement_texture,
  displacement_scale = 1,
  use_cube = FALSE,
  cube_subdivision_levels = NA,
  displace = TRUE,
  verbose = TRUE,
  position = c(0, 0, 0),
  scale = c(1, 1, 1),
  angle = c(0, 0, 0),
  pivot_point = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  material = material_list()
)
```

Arguments

displacement_texture	Image or matrix/array that will be used to displace the sphere.
displacement_scale	Default 1. Scale of the displacement.
use_cube	Default FALSE. Whether to use a subdivided cube instead of a UV sphere. Use this if you want to visualize areas near the poles.
cube_subdivision_levels	Default NA. Uses the dimensions of the displacement texture to automatically calculate the number of subdivision levels.
displace	Default TRUE. Whether to displace the sphere, or just generate the initial mesh for later displacement.
verbose	Default TRUE. Whether to print displacement texture information.
position	Default c(0,0,0). Position of the mesh.
scale	Default c(1,1,1). Scale of the mesh. Can also be a single numeric value scaling all axes uniformly.
angle	Default c(0,0,0). Angle to rotate the mesh.
pivot_point	Default c(0,0,0). Point around which to rotate the mesh.
order_rotation	Default c(1,2,3). Order to rotate the axes.
material	Default material_list() (default values). Specify the material of the object.

Value

raymesh object

Examples

```
if(run_documentation()) {

}
```

`displace_mesh` *Displace a Mesh*

Description

Displace a Mesh

Usage

```
displace_mesh(
  mesh,
  displacement_texture,
  displacement_scale = 1,
  displacement_vector = FALSE,
  id = NA,
  verbose = TRUE
)
```

Arguments

`mesh`	The mesh.
`displacement_texture`	Image or matrix/array that will be used to displace the mesh
`displacement_scale`	Default `1`. Intensity of the displacement effect. Higher values result in greater displacement.
`displacement_vector`	Default `FALSE`. Whether to use vector displacement. If `TRUE`, the displacement texture is interpreted as providing a 3D displacement vector. Otherwise, the texture is interpreted as providing a scalar displacement.
`id`	Default `NA` (all shapes). The shape index to have new normals calculated.
`verbose`	Default `TRUE`. Whether to print displacement texture information.

Value

raymesh object

Examples

```
if(run_documentation()) {
 #Let's construct a mesh from the volcano dataset
}
```

---

flip_orientation_mesh *Flip Orientation*

Description

Flip Orientation

Usage

```
flip_orientation_mesh(mesh)
```

Arguments

mesh	The mesh to swap orientations.

Value

Mesh with flipped vertex orientation

Examples

```
# Flip a mesh
if(run_documentation()) {
sphere_mesh(position=c(-1,0,0)) |>
  add_shape(flip_orientation_mesh(sphere_mesh(position=c(1,0,0)))) |>
  rasterize_scene(debug="normals",fov=30)
}
```

---

generate_cornell_mesh *Cornell Box 3D Model*

Description

Cornell Box 3D Model

Usage

```
generate_cornell_mesh(
  leftcolor = "#1f7326",
  rightcolor = "#a60d0d",
  roomcolor = "#bababa",
  ceiling = TRUE,
  light = TRUE
)
```

Arguments

leftcolor	Default #1f7326 (green).
rightcolor	Default #a60d0d (red).
roomcolor	Default #bababa (light grey).
ceiling	Default TRUE. Whether to render the ceiling.
light	Default TRUE. Whether to render a point light near the ceiling.

Value

List describing the mesh.

Examples

```
if(run_documentation()) {
#Generate and render the default Cornell box and add an object.
generate_cornell_mesh() |>
  rasterize_scene()
}
if(run_documentation()) {
#Add an object to the scene
generate_cornell_mesh() |>
 add_shape(obj_mesh(r_obj(),position=c(555/2,555/2,555/2),scale=300,angle=c(0,180,0))) |>
  rasterize_scene()
}
if(run_documentation()) {
#Turn off the ceiling so the default directional light reaches inside the box
generate_cornell_mesh(ceiling=FALSE) |>
 add_shape(obj_mesh(r_obj(),position=c(555/2,555/2,555/2),scale=300,angle=c(0,180,0))) |>
  rasterize_scene()
}
if(run_documentation()) {
#Adjust the light to the front
generate_cornell_mesh(ceiling=FALSE) |>
 add_shape(obj_mesh(r_obj(),position=c(555/2,555/2,555/2),scale=300,angle=c(0,180,0))) |>
  rasterize_scene(light_info = directional_light(direction=c(0,1,-1)))
  }
if(run_documentation()) {
#Change the color palette
generate_cornell_mesh(ceiling=FALSE,leftcolor="purple", rightcolor="yellow") |>
 add_shape(obj_mesh(r_obj(),position=c(555/2,555/2,555/2),scale=300,angle=c(0,180,0))) |>
  rasterize_scene(light_info = directional_light(direction=c(0,1,-1)))
}
```

---

generate_line *Generate Lines*

Description

Generate Lines

Usage

```
generate_line(start = c(0, 0, 0), end = c(0, 1, 0), color = "white")
```

Arguments

start	Default `c(0,0,0)`. Start of the line segment.
end	Default `c(0,1,0)`. End of the line segment..
color	Default `white`. Color of the line segment.

Value

Line matrix

Examples

```
if(run_documentation()) {
# Make a spiral of lines
t = seq(0,8*pi,length.out=361)
line_mat = matrix(nrow=0,ncol=9)

for(i in 1:360) {
  line_mat = add_lines(line_mat,
                    generate_line(start = c(0.5*sin(t[i]), t[i]/(8*pi), 0.5*cos(t[i])),
                             end  = c(0.5*sin(t[i+1]), t[i+1]/(8*pi), 0.5*cos(t[i+1]))))
}
rasterize_lines(line_mat)
}
if(run_documentation()) {
#Change the line color
line_mat = matrix(nrow=0,ncol=9)
cols = hsv(seq(0,1,length.out=360))
for(i in 1:360) {
  line_mat = add_lines(line_mat,
                     generate_line(start = c(sin(t[i]), 2*t[i]/(8*pi), cos(t[i])),
                                 end  = c(sin(t[i+1]), 2*t[i+1]/(8*pi), cos(t[i+1])),
                                 color = cols[i]))
}
rasterize_lines(line_mat,lookfrom=c(0,10,10),fov=15)
}
if(run_documentation()) {
#Use in a scene with a mesh
obj_mesh(r_obj(simple_r = TRUE),material=material_list(diffuse="dodgerblue")) |>
 rasterize_scene(line_info = line_mat, light_info = directional_light(c(0,1,1)),
                 lookfrom=c(0,5,10),lookat=c(0,0.8,0),fov=15)
}
```

get_mesh_bbox *Get Mesh Bounding Box*

Description

Calculates the bounding box of a mesh

Usage

```
get_mesh_bbox(mesh)
```

Arguments

mesh	The mesh object.

Value

2x3 numeric matrix

Examples

```
if(run_documentation()) {
#Calculates the center of the mesh
get_mesh_bbox(generate_cornell_mesh())
}
```

get_mesh_center *Get Mesh Center*

Description

Calculates the coordinates of the center of a mesh

Usage

```
get_mesh_center(mesh)
```

Arguments

mesh	The mesh object.

Value

Length-3 numeric vector

Examples

```
if(run_documentation()) {
#Calculates the center of the mesh
get_mesh_center(generate_cornell_mesh())
}
```

`material_list` *Material List*

Description

Generate a material properties list.

Usage

```
material_list(
  diffuse = c(0.8, 0.8, 0.8),
  ambient = c(0, 0, 0),
  specular = c(1, 1, 1),
  transmittance = c(0, 0, 0),
  emission = c(0, 0, 0),
  shininess = 50,
  ior = 1,
  dissolve = 1,
  illum = 1,
  texture_location = "",
  normal_texture_location = "",
  bump_texture_location = "",
  specular_texture_location = "",
  ambient_texture_location = "",
  emissive_texture_location = "",
  diffuse_intensity = 1,
  bump_intensity = 1,
  specular_intensity = 1,
  emission_intensity = 1,
  ambient_intensity = 1,
  culling = "back",
  type = "diffuse",
  translucent = TRUE,
  toon_levels = 5,
  toon_outline_width = 0.05,
  toon_outline_color = "black",
  reflection_intensity = 0,
  reflection_sharpness = 1,
  two_sided = FALSE
)
```

Arguments

diffuse	Default c(0.5,0.5,0.5). The diffuse color.
ambient	Default c(0,0,0). The ambient color.
specular	Default c(1,1,1). The specular color.
transmittance	Default c(0,0,0). The transmittance
emission	Default c(0,0,0). The emissive color.
shininess	Default 50.0. The shininess exponent.
ior	Default 1.0. The index of refraction. If this is not equal to 1.0, the material will be refractive.
dissolve	Default 1.0. The transparency.
illum	Default 1.0. The illumination.
texture_location	Default "". The diffuse texture location.
normal_texture_location	Default "". The normal texture location.
bump_texture_location	Default "". The bump texture location.
specular_texture_location	Default "". The specular texture location.
ambient_texture_location	Default "". The ambient texture location.
emissive_texture_location	Default "". The emissive texture location.
diffuse_intensity	Default 1. The diffuse intensity.
bump_intensity	Default 1. The bump intensity.
specular_intensity	Default 1. The specular intensity.
emission_intensity	Default 1. The emission intensity.
ambient_intensity	Default 1. The ambient intensity.
culling	Default "back". The culling type. Options are back, front, and none.
type	Default "diffuse". The shader type. Options include diffuse,phong,vertex, and color.
translucent	Default FALSE. Whether light should transmit through a semi-transparent material.
toon_levels	Default 5. Number of color breaks in the toon shader.
toon_outline_width	Default 0.05. Expansion term for model to specify toon outline width.
toon_outline_color	Default black. Toon outline color.

`reflection_intensity`
: Default `0.0`. Intensity of the reflection of the environment map, if present. This will be ignored if the material is refractive.

`reflection_sharpness`
: Default `1.0`. Sharpness of the reflection, where lower values have blurrier reflections. Must be greater than zero and less than one.

`two_sided`
: Default `FALSE`. Whether diffuse materials should be two sided (normal is taken as the absolute value of the dot product of the light direction and the normal).

Value

List of material properties.

Examples

```
if(run_documentation()) {
mat_prop = material_list(diffuse="purple", type="phong", shininess = 20,
                         ambient="purple", ambient_intensity=0.3,
                         specular = "red", specular_intensity=2)

p_sphere = sphere_mesh(position=c(555/2,555/2,555/2),
                       radius=40,material=mat_prop)

rasterize_scene(p_sphere, light_info=directional_light(direction=c(0.1,0.6,-1)))
}
```

---

mesh3d_mesh *Mesh3d 3D Model*

Description

Mesh3d 3D Model

Usage

```
mesh3d_mesh(
  mesh,
  center = FALSE,
  position = c(0, 0, 0),
  scale = c(1, 1, 1),
  angle = c(0, 0, 0),
  pivot_point = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  materialspath = NULL,
  material = material_list()
)
```

Arguments

mesh	Mesh3d object.
center	Default FALSE. Whether to center the mesh.
position	Default c(0,0,0). Position of the mesh.
scale	Default c(1,1,1). Scale of the mesh. Can also be a single numeric value scaling all axes uniformly.
angle	Default c(0,0,0). Angle to rotate the mesh.
pivot_point	Default c(0,0,0). Point around which to rotate the mesh.
order_rotation	Default c(1,2,3). Order to rotate the axes.
materialspath	Default NULL. Path to the MTL file, if different from the OBJ file.
material	Default NULL, read from the MTL file. If not NULL, this accepts the output from the material_list() function to specify the material.

Value

List describing the mesh.

Examples

```
if(run_documentation()) {
  # Read in a mesh3d object and rasterize it
  library(Rvcg)
  data(humface)

  mesh3d_mesh(humface,position = c(0,-0.3,0),scale = 1/70,
              material=material_list(diffuse="dodgerblue4", type="phong", shininess=20,
              ambient = "dodgerblue4", ambient_intensity=0.3)) |>
    rasterize_scene(lookat = c(0,0.5,1), light_info = directional_light(c(1,0.5,1)))
 }

 if(run_documentation()) {
  # Subdivide the mesh for a smoother appearance
  mesh3d_mesh(humface,position = c(0,-0.3,0),scale = 1/70,
              material=material_list(diffuse="dodgerblue4", type="phong", shininess=20,
              ambient = "dodgerblue4", ambient_intensity=0.3)) |>
    subdivide_mesh() |>
    rasterize_scene(lookat = c(0,0.5,1), light_info = directional_light(c(1,0.5,1)))
 }
```

---

obj_mesh *OBJ Mesh 3D Model*

Description

OBJ Mesh 3D Model

Usage

```
obj_mesh(
  filename,
  position = c(0, 0, 0),
  scale = c(1, 1, 1),
  angle = c(0, 0, 0),
  pivot_point = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  materialspath = NULL,
  center = FALSE,
  material = NULL
)
```

Arguments

`filename`	OBJ filename.
`position`	Default `c(0,0,0)`. Position of the mesh.
`scale`	Default `c(1,1,1)`. Scale of the mesh. Can also be a single numeric value scaling all axes uniformly.
`angle`	Default `c(0,0,0)`. Angle to rotate the mesh.
`pivot_point`	Default `c(0,0,0)`. Point around which to rotate the mesh.
`order_rotation`	Default `c(1,2,3)`. Order to rotate the axes.
`materialspath`	Default `NULL`. Path to the MTL file, if different from the OBJ file.
`center`	Default `FALSE`. Whether to center the mesh.
`material`	Default `NULL`, read from the MTL file. If not `NULL`, this accepts the output from the `material_list()` function to specify the material.

Value

List describing the mesh.

Examples

```
if(run_documentation()) {
#Read in the provided 3D R mesh
generate_cornell_mesh(ceiling=FALSE) |>
 add_shape(obj_mesh(r_obj(),position=c(555/2,555/2,555/2),scale=400,angle=c(0,180,0))) |>
  rasterize_scene(light_info = directional_light(direction=c(0.2,0.5,-1)))
}
```

`ply_mesh` *PLY Mesh 3D Model*

Description

PLY Mesh 3D Model

Usage

```
ply_mesh(
  filename,
  center = FALSE,
  position = c(0, 0, 0),
  scale = c(1, 1, 1),
  angle = c(0, 0, 0),
  pivot_point = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  material = material_list()
)
```

Arguments

`filename`	PLY filename.
`center`	Default `FALSE`. Whether to center the mesh.
`position`	Default `c(0,0,0)`. Position of the mesh.
`scale`	Default `c(1,1,1)`. Scale of the mesh. Can also be a single numeric value scaling all axes uniformly.
`angle`	Default `c(0,0,0)`. Angle to rotate the mesh.
`pivot_point`	Default `c(0,0,0)`. Point around which to rotate the mesh.
`order_rotation`	Default `c(1,2,3)`. Order to rotate the axes.
`material`	Default `material_list()` (default values). Specify the material of the object.

Value

List describing the mesh.

Examples

```
#See the documentation for `obj_mesh()`--no example PLY models are included with this package,
#but the process of loading a model is the same (but no materials are included in PLY files).
```

point_light *Point light*

Description

The falloff of the point light intensity is given by the following equation (referenc:

Intensity = intensity / (constant + falloff * distance + falloff_quad * (distance * distance));

Usage

```
point_light(
  position = c(0, 0, 0),
  color = "white",
  intensity = 1,
  constant = 1,
  falloff = 1,
  falloff_quad = 1
)
```

Arguments

position	A two-dimensional matrix, where each entry in the matrix is the elevation at that point. All points are assumed to be evenly spaced.
color	Default 400. Width of the rendered image.
intensity	Default 1. Intensity of the point light.
constant	Default 1. Constant term. See description for details.
falloff	Default 1. Linear falloff term. See description for details.
falloff_quad	Default 1. Quadratic falloff term. See description for details.

Value

A matrix representing the light information.

Examples

```
if(run_documentation()) {
#Add point lights and vary the intensity
lights_int = point_light(position=c(100,100,400), color="white", intensity=0.125,
                     falloff_quad = 0.0, constant = 0.0002, falloff = 0.005) |>
 add_light(point_light(position=c(100,455,400), color="white", intensity=0.25,
                      falloff_quad = 0.0, constant = 0.0002, falloff = 0.005)) |>
 add_light(point_light(position=c(455,100,400), color="white", intensity=0.5,
                      falloff_quad = 0.0, constant = 0.0002, falloff = 0.005)) |>
 add_light(point_light(position=c(455,455,400), color="white", intensity=1,
                      falloff_quad = 0.0, constant = 0.0002, falloff = 0.005))
```

```
generate_cornell_mesh(light=FALSE) |>
 rasterize_scene(light_info = lights_int)

#Add point lights and vary the color
lights_c = point_light(position=c(100,100,500), color="red",
                       falloff_quad = 0.0, constant = 0.0002, falloff = 0.005) |>
 add_light(point_light(position=c(100,455,500), color="blue",
                       falloff_quad = 0.0, constant = 0.0002, falloff = 0.005)) |>
 add_light(point_light(position=c(455,100,500), color="purple",
                       falloff_quad = 0.0, constant = 0.0002, falloff = 0.005)) |>
 add_light(point_light(position=c(455,455,500), color="yellow",
                       falloff_quad = 0.0, constant = 0.0002, falloff = 0.005))

generate_cornell_mesh(light=FALSE) |>
 rasterize_scene(light_info = lights_c)

#Add point lights and vary the falloff term
lights_fo = point_light(position=c(100,100,500), color="white",
                       falloff_quad = 0.0, constant = 0.0002, falloff = 0.005) |>
 add_light(point_light(position=c(100,455,500), color="white",
                       falloff_quad = 0.0, constant = 0.0002, falloff = 0.01)) |>
 add_light(point_light(position=c(455,100,500), color="white",
                       falloff_quad = 0.0, constant = 0.0002, falloff = 0.02)) |>
 add_light(point_light(position=c(455,455,500), color="white",
                       falloff_quad = 0.0, constant = 0.0002, falloff = 0.04))

generate_cornell_mesh(light=FALSE) |>
 rasterize_scene(light_info = lights_fo)

#Add point lights and vary the quadradic falloff term
lights_quad = point_light(position=c(100,100,500), color="white",
                       falloff_quad = 0.0001, constant = 0.0002, falloff = 0.005) |>
 add_light(point_light(position=c(100,455,500), color="white",
                       falloff_quad = 0.0002, constant = 0.0002, falloff = 0.005)) |>
 add_light(point_light(position=c(455,100,500), color="white",
                       falloff_quad = 0.0004, constant = 0.0002, falloff = 0.005)) |>
 add_light(point_light(position=c(455,455,500), color="white",
                       falloff_quad = 0.0008, constant = 0.0002, falloff = 0.005))

generate_cornell_mesh(light=FALSE) |>
 rasterize_scene(light_info = lights_quad)
}
```

`rasterize_lines` *Rasterize Lines*

Description

Render a 3D scene made out of lines using a software rasterizer.

Usage

```
rasterize_lines(
  line_info = NULL,
  filename = NA,
  width = 800,
  height = 800,
  alpha_line = 1,
  parallel = TRUE,
  fov = 20,
  lookfrom = c(0, 0, 10),
  lookat = NULL,
  camera_up = c(0, 1, 0),
  color = "red",
  background = "black",
  debug = "none",
  near_plane = 0.1,
  far_plane = 100,
  block_size = 4,
  ortho_dimensions = c(1, 1),
  bloom = FALSE,
  antialias_lines = TRUE
)
```

Arguments

line_info	The mesh object.
filename	Default NULL. Filename to save the image. If NULL, the image will be plotted.
width	Default 400. Width of the rendered image.
height	Default 400. Width of the rendered image.
alpha_line	Default 1. Line transparency.
parallel	Default TRUE. Whether to use parallel processing.
fov	Default 20. Width of the rendered image.
lookfrom	Default c(0,0,10). Camera location.
lookat	Default NULL. Camera focal position, defaults to the center of the model.
camera_up	Default c(0,1,0). Camera up vector.
color	Default darkred. Color of model if no material file present (or for faces using the default material).
background	Default white. Background color.
debug	Default "none".
near_plane	Default 0.1.
far_plane	Default 100.
block_size	Default 4.

`ortho_dimensions`
: Default `c(1,1)`. Width and height of the orthographic camera. Will only be used if `fov = 0`.

`bloom`
: Default `FALSE`. Whether to apply bloom to the image. If `TRUE`, this performs a convolution of the HDR image of the scene with a sharp, long-tailed exponential kernel, which does not visibly affect dimly pixels, but does result in emitters light slightly bleeding into adjacent pixels.

`antialias_lines`
: Default `TRUE`. Whether to anti-alias lines in the scene.

Value

Rasterized image.

Examples

```
if(run_documentation()) {
#Generate a cube out of lines
cube_outline = generate_line(start = c(-1, -1, -1), end = c(-1, -1, 1)) |>
  add_lines(generate_line(start = c(-1, -1, -1), end = c(-1, 1, -1))) |>
  add_lines(generate_line(start = c(-1, -1, -1), end = c(1, -1, -1))) |>
  add_lines(generate_line(start = c(-1, -1, 1), end = c(-1, 1, 1))) |>
  add_lines(generate_line(start = c(-1, -1, 1), end = c(1, -1, 1))) |>
  add_lines(generate_line(start = c(-1, 1, 1), end = c(-1, 1, -1))) |>
  add_lines(generate_line(start = c(-1, 1, 1), end = c(1, 1, 1))) |>
  add_lines(generate_line(start = c(1, 1, -1), end = c(1, -1, -1))) |>
  add_lines(generate_line(start = c(1, 1, -1), end = c(1, 1, 1))) |>
  add_lines(generate_line(start = c(1, -1, -1), end = c(1, -1, 1))) |>
  add_lines(generate_line(start = c(1, -1, 1), end = c(1, 1, 1))) |>
  add_lines(generate_line(start = c(-1, 1, -1), end = c(1, 1, -1)))
rasterize_lines(cube_outline,fov=90,lookfrom=c(0,0,3))
}
if(run_documentation()) {
#Scale the cube uniformly
scaled_cube = color_lines(scale_lines(cube_outline,scale=0.5),color="red")
rasterize_lines(add_lines(cube_outline,scaled_cube),fov=90,lookfrom=c(0,0,3))
}
if(run_documentation()) {
#Scale the cube non-uniformly
scaled_cube = color_lines(scale_lines(cube_outline,scale=c(0.8,2,0.4)),color="red")
rasterize_lines(add_lines(cube_outline,scaled_cube),fov=60,lookfrom=c(3,3,3))
}
```

---

rasterize_scene *Rasterize Scene*

Description

Render a 3D scene with meshes, lights, and lines using a software rasterizer.

Usage

```
rasterize_scene(
  scene,
  filename = NA,
  width = 800,
  height = 800,
  line_info = NULL,
  alpha_line = 1,
  parallel = TRUE,
  plot = is.na(filename),
  fov = 20,
  lookfrom = c(0, 0, 10),
  lookat = NULL,
  camera_up = c(0, 1, 0),
  fsaa = 2,
  light_info = directional_light(),
  color = "red",
  type = "diffuse",
  background = "black",
  tangent_space_normals = TRUE,
  shadow_map = TRUE,
  shadow_map_bias = 0.003,
  shadow_map_intensity = 0,
  shadow_map_dims = NULL,
  ssao = FALSE,
  ssao_intensity = 10,
  ssao_radius = 0.1,
  tonemap = "none",
  debug = "none",
  near_plane = 0.1,
  far_plane = 100,
  shader = "default",
  block_size = 4,
  shape = NULL,
  line_offset = 1e-05,
  ortho_dimensions = c(1, 1),
  bloom = FALSE,
  antialias_lines = TRUE,
  environment_map = "",
  background_sharpness = 1,
  verbose = FALSE,
  vertex_transform = NULL,
  validate_scene = TRUE
)
```

Arguments

scene	The scene object.

filename	Default NULL. Filename to save the image. If NULL, the image will be plotted.
width	Default 400. Width of the rendered image.
height	Default 400. Width of the rendered image.
line_info	Default NULL. Matrix of line segments to add to the scene. Number of rows must be a multiple of 2.
alpha_line	Default 1. Line transparency.
parallel	Default TRUE. Whether to use parallel processing.
plot	Default is.na(filename). Whether to plot the image.
fov	Default 20. Width of the rendered image.
lookfrom	Default c(0,0,10). Camera location.
lookat	Default NULL. Camera focal position, defaults to the center of the model.
camera_up	Default c(0,1,0). Camera up vector.
fsaa	Default 2. Full screen anti-aliasing multiplier. Must be positive integer, higher numbers will improve anti-aliasing quality but will vastly increase memory usage.
light_info	Default directional_light(). Description of scene lights, generated with the point_light() and directional_light() functions.
color	Default darkred. Color of model if no material file present (or for faces using the default material).
type	Default diffuse. Shader type. Other options: vertex (Gouraud shading), phong, and color (no lighting).
background	Default white. Background color.
tangent_space_normals	Default TRUE.
shadow_map	Default FALSE.
shadow_map_bias	Default 0.005.
shadow_map_intensity	Default 0.5.
shadow_map_dims	Default NULL.
ssao	Default FALSE. Whether to add screen-space ambient occlusion (SSAO) to the render.
ssao_intensity	Default 10. Intensity of the shadow map.
ssao_radius	Default 0.1. Radius to use when calculating the SSAO term.
tonemap	Default "none".
debug	Default "none".
near_plane	Default 0.1.
far_plane	Default 100.
shader	Default "default".

`block_size` Default `4`.

`shape` Default `NULL`. The shape to render in the OBJ mesh.

`line_offset` Default `0.0001`. Amount to offset lines towards camera to prevent z-fighting.

`ortho_dimensions`
Default `c(1,1)`. Width and height of the orthographic camera. Will only be used if `fov = 0`.

`bloom` Default `FALSE`. Whether to apply bloom to the image. If `TRUE`, this performs a convolution of the HDR image of the scene with a sharp, long-tailed exponential kernel, which does not visibly affect dimly pixels, but does result in emitters light slightly bleeding into adjacent pixels.

`antialias_lines`
Default `TRUE`. Whether to anti-alias lines in the scene.

`environment_map`
Default `""`. Image file to use as a texture for all reflective and refractive materials in the scene, along with the background.

`background_sharpness`
Default `1.0`. A number greater than zero but less than one indicating the sharpness of the background image.

`verbose` Default `FALSE`. Prints out timing information.

`vertex_transform`
Default `NULL`. A function that transforms the vertex locations, based on their location. Function should takes a length-3 numeric vector and returns another length-3 numeric vector as the output.

`validate_scene` Default `TRUE`. Whether to validate the scene input.

Value

Rasterized image.

Examples

```
if(run_documentation()) {
#Let's load the cube OBJ file included with the package

rasterize_scene(cube_mesh(),lookfrom=c(2,4,10),
              light_info = directional_light(direction=c(0.5,1,0.7)))
}
if(run_documentation()) {
#Flatten the cube, translate downwards, and set to grey
base_model = cube_mesh() |>
 scale_mesh(scale=c(5,0.2,5)) |>
 translate_mesh(c(0,-0.1,0)) |>
 set_material(diffuse="grey80")

rasterize_scene(base_model, lookfrom=c(2,4,10),
              light_info = directional_light(direction=c(0.5,1,0.7)))
}
if(run_documentation()) {
```

```
#load the R OBJ file, scale it down, color it blue, and add it to the grey base
r_model = obj_mesh(r_obj(simple_r = TRUE)) |>
 scale_mesh(scale=0.5) |>
 set_material(diffuse="dodgerblue") |>
 add_shape(base_model)

rasterize_scene(r_model, lookfrom=c(2,4,10),
              light_info = directional_light(direction=c(0.5,1,0.7)))
}
if(run_documentation()) {
#Zoom in and reduce the shadow mapping intensity
rasterize_scene(r_model, lookfrom=c(2,4,10), fov=10,shadow_map = TRUE, shadow_map_intensity=0.3,
              light_info = directional_light(direction=c(0.5,1,0.7)))
}
if(run_documentation()) {
#Include the resolution (4x) of the shadow map for less pixellation around the edges
#Also decrease the shadow_map_bias slightly to remove the "peter panning" floating shadow effect
rasterize_scene(r_model, lookfrom=c(2,4,10), fov=10,
              shadow_map_dims=4,
              light_info = directional_light(direction=c(0.5,1,0.7)))
}
if(run_documentation()) {
#Add some more directional lights and change their color
lights = directional_light(c(0.7,1.1,-0.9),color = "orange",intensity = 1) |>
          add_light(directional_light(c(0.7,1,1),color = "dodgerblue",intensity = 1)) |>
           add_light(directional_light(c(2,4,10),color = "white",intensity = 0.5))
rasterize_scene(r_model, lookfrom=c(2,4,10), fov=10,
              light_info = lights)
}
if(run_documentation()) {
#Add some point lights
lights_p = lights |>
 add_light(point_light(position=c(-1,1,0),color="red", intensity=2)) |>
 add_light(point_light(position=c(1,1,0),color="purple", intensity=2))
rasterize_scene(r_model, lookfrom=c(2,4,10), fov=10,
              light_info = lights_p)
}
if(run_documentation()) {
#change the camera position
rasterize_scene(r_model, lookfrom=c(-2,2,-10), fov=10,
              light_info = lights_p)
}
if(run_documentation()) {

#Add a spiral of lines around the model by generating a matrix of line segments
t = seq(0,8*pi,length.out=361)
line_mat = matrix(nrow=0,ncol=9)

for(i in 1:360) {
  line_mat = add_lines(line_mat,
                    generate_line(start = c(0.5*sin(t[i]), t[i]/(8*pi), 0.5*cos(t[i])),
                             end  = c(0.5*sin(t[i+1]), t[i+1]/(8*pi), 0.5*cos(t[i+1]))))
}
```

```
rasterize_scene(r_model, lookfrom=c(2,4,10), fov=10, line_info = line_mat,
              light_info = lights)
}
```

`read_obj` *Load an OBJ file*

Description

Loads an OBJ file and return a `ray_mesh` list structure. No processing is done on the object other than loading it (unlike `obj_model()`).

Usage

```
read_obj(filename, materialspath = NULL)
```

Arguments

`filename`	Filename of the OBJ file.
`materialspath`	Directory where the MTL file is located. Defaults to the directory of `filename`.

Value

`ray_mesh` list object #Load an arrow OBJ sphere = read_obj(system.file("extdata", "arrow.txt", package="rayvertex"))

`rotate_lines` *Rotate Lines*

Description

Rotate Lines

Usage

```
rotate_lines(
  lines,
  angle = c(0, 0, 0),
  pivot_point = c(0, 0, 0),
  order_rotation = c(1, 2, 3)
)
```

Arguments

`lines`	The existing line scene.
`angle`	Default `c(0,0,0)`. The rotation amount for the x/y/z axes, in degrees.
`pivot_point`	Default `c(0,0,0)`. The pivot point of the rotation.
`order_rotation`	Default `c(1,2,3)`. The order in which to perform the rotations.#'

Value

Rotated lines.

Examples

```
if(run_documentation()) {
#Generate a cube out of lines
cube_outline = generate_line(start = c(-1, -1, -1), end = c(-1, -1, 1)) |>
  add_lines(generate_line(start = c(-1, -1, -1), end = c(-1, 1, -1))) |>
  add_lines(generate_line(start = c(-1, -1, -1), end = c(1, -1, -1))) |>
  add_lines(generate_line(start = c(-1, -1, 1), end = c(-1, 1, 1))) |>
  add_lines(generate_line(start = c(-1, -1, 1), end = c(1, -1, 1))) |>
  add_lines(generate_line(start = c(-1, 1, 1), end = c(-1, 1, -1))) |>
  add_lines(generate_line(start = c(-1, 1, 1), end = c(1, 1, 1))) |>
  add_lines(generate_line(start = c(1, 1, -1), end = c(1, -1, -1))) |>
  add_lines(generate_line(start = c(1, 1, -1), end = c(1, 1, 1))) |>
  add_lines(generate_line(start = c(1, -1, -1), end = c(1, -1, 1))) |>
  add_lines(generate_line(start = c(1, -1, 1), end = c(1, 1, 1))) |>
  add_lines(generate_line(start = c(-1, 1, -1), end = c(1, 1, -1)))
rasterize_lines(cube_outline,lookfrom=c(0,6,10))
}
if(run_documentation()) {
#Rotate the cube 30 degrees around the y-axis
rotated_cube = color_lines(rotate_lines(cube_outline,angle=c(0,30,0)),color="red")
rasterize_lines(add_lines(cube_outline,rotated_cube),lookfrom=c(0,6,10))
}
if(run_documentation()) {
#Rotate the cube 30 degrees around each axis, in this order: x,y,z
rotated_cube = color_lines(rotate_lines(cube_outline,angle=c(30,30,30)),color="red")
rasterize_lines(add_lines(cube_outline,rotated_cube),lookfrom=c(0,6,10))
}
if(run_documentation()) {
#Rotate the cube 30 degrees around each axis, in this order: z,y,x
rotated_cube = color_lines(rotate_lines(cube_outline,angle=c(30,30,30),
                           order_rotation = c(3,2,1)),color="red")
rasterize_lines(add_lines(cube_outline,rotated_cube),lookfrom=c(0,6,10))
}
```

rotate_mesh *Rotate Mesh*

Description

Rotate Mesh

Usage

```
rotate_mesh(
  mesh,
  angle = c(0, 0, 0),
  pivot_point = c(0, 0, 0),
  order_rotation = c(1, 2, 3)
)
```

Arguments

mesh	The mesh.
angle	Default `c(0,0,0)`. The rotation amount for the x/y/z axes, in degrees.
pivot_point	Default `c(0,0,0)`. The pivot point of the rotation.
order_rotation	Default `c(1,2,3)`. The order in which to perform the rotations.

Value

Rotated Mesh

Examples

```
if(run_documentation()) {
#Rotate a mesh in the Cornell box
robj = obj_mesh(r_obj(), scale=150,angle=c(0,180,0))

generate_cornell_mesh() |>
add_shape(rotate_mesh(translate_mesh(robj,c(400,100,155)),c(0,30,0),
                      pivot_point=c(400,100,155))) |>
add_shape(rotate_mesh(translate_mesh(robj,c(555/2,200,555/2)),c(-30,60,30),
                      pivot_point=c(555/2,200,555/2))) |>
add_shape(rotate_mesh(translate_mesh(robj,c(155,300,400)),c(-30,60,30),
                      pivot_point=c(155,300,400), order_rotation=c(3,2,1))) |>
rasterize_scene(light_info=directional_light(direction=c(0.1,0.6,-1)))
}
```

run_documentation *Run Documentation*

Description

This function determines if the examples are being run in pkgdown. It is not meant to be called by the user.

Usage

```
run_documentation()
```

Value

Boolean value.

Examples

```
# See if the documentation should be run.
run_documentation()
```

r_obj *R 3D Model*

Description

3D obj model of R logo (created from the R SVG logo with the `raybevel` package), to be used with `obj_model()`

Usage

```
r_obj(simple_r = FALSE)
```

Arguments

`simple_r`	Default `FALSE`. If `TRUE`, this will return a 3D R (instead of the R logo).

Value

File location of the 3d_r_logo.obj file (saved with a .txt extension)

Examples

```
#Load and render the included example R object file.
if(run_documentation()) {
obj_mesh(r_obj()) |>
  rasterize_scene(lookfrom = c(0, 1, 10),
                  fov=7,light_info = directional_light(c(1,1,1)))
}
```

scale_lines *Scale Lines*

Description

Scale Lines

Usage

```
scale_lines(lines, scale = 1)
```

Arguments

lines	The line scene.
scale	Default c(1,1,1). The scale amount, per axis.

Value

Scaled line matrix.

Examples

```
if(run_documentation()) {
#Generate a cube out of lines
cube_outline = generate_line(start = c(-1, -1, -1), end = c(-1, -1, 1)) |>
  add_lines(generate_line(start = c(-1, -1, -1), end = c(-1, 1, -1))) |>
  add_lines(generate_line(start = c(-1, -1, -1), end = c(1, -1, -1))) |>
  add_lines(generate_line(start = c(-1, -1, 1), end = c(-1, 1, 1))) |>
  add_lines(generate_line(start = c(-1, -1, 1), end = c(1, -1, 1))) |>
  add_lines(generate_line(start = c(-1, 1, 1), end = c(-1, 1, -1))) |>
  add_lines(generate_line(start = c(-1, 1, 1), end = c(1, 1, 1))) |>
  add_lines(generate_line(start = c(1, 1, -1), end = c(1, -1, -1))) |>
  add_lines(generate_line(start = c(1, 1, -1), end = c(1, 1, 1))) |>
  add_lines(generate_line(start = c(1, -1, -1), end = c(1, -1, 1))) |>
  add_lines(generate_line(start = c(1, -1, 1), end = c(1, 1, 1))) |>
  add_lines(generate_line(start = c(-1, 1, -1), end = c(1, 1, -1)))
rasterize_lines(cube_outline,fov=90,lookfrom=c(0,0,3))
}
if(run_documentation()) {
#Scale the cube uniformly
scaled_cube = color_lines(scale_lines(cube_outline,scale=0.5),color="red")
rasterize_lines(add_lines(cube_outline,scaled_cube),fov=90,lookfrom=c(0,0,3))
}
if(run_documentation()) {
#Scale the cube non-uniformly
scaled_cube = color_lines(scale_lines(cube_outline,scale=c(0.8,2,0.4)),color="red")
rasterize_lines(add_lines(cube_outline,scaled_cube),fov=60,lookfrom=c(3,3,3))
}
```

scale_mesh *Scale Mesh*

Description

Scale Mesh

Usage

```
scale_mesh(mesh, scale = 1, center = c(0, 0, 0))
```

Arguments

mesh	The mesh.
scale	Default c(1,1,1). The scale amount, per axis.
center	Default c(0,0,0). The center of the scale.

Value

Scaled mesh

Examples

```
if(run_documentation()) {
#Scale a mesh in the Cornell box
robj = obj_mesh(r_obj(), scale=150,angle=c(0,180,0))

generate_cornell_mesh() |>
add_shape(scale_mesh(translate_mesh(robj,c(400,100,155)),0.5, center=c(400,100,155))) |>
add_shape(scale_mesh(translate_mesh(robj,c(555/2,200,555/2)),1.5, center=c(555/2,200,555/2))) |>
add_shape(scale_mesh(translate_mesh(robj,c(55,300,400)),c(0.5,2,0.5), center=c(155,300,400))) |>
rasterize_scene(light_info=directional_light(direction=c(0.1,0.6,-1)))
}
```

scale_unit_mesh *Scale Mesh to Unit Bounding Box*

Description

Scale Mesh to Unit Bounding Box

Usage

```
scale_unit_mesh(mesh, center_mesh = FALSE)
```

Arguments

mesh	The mesh.
center_mesh	Default FALSE. Whether to center the mesh at the origin after scaling.

Value

Scaled mesh

Examples

```
if(run_documentation()) {
#Scale the Cornell box (and contents) down to the unit box.
robj = obj_mesh(r_obj(), scale=150,angle=c(0,180,0))

generate_cornell_mesh() |>
add_shape(scale_mesh(translate_mesh(robj,c(400,100,155)),0.5, center=c(400,100,155))) |>
add_shape(scale_mesh(translate_mesh(robj,c(555/2,200,555/2)),1.5, center=c(555/2,200,555/2))) |>
add_shape(scale_mesh(translate_mesh(robj,c(55,300,400)),c(0.5,2,0.5), center=c(155,300,400))) |>
scale_unit_mesh(center_mesh = TRUE) |>
rasterize_scene(light_info=directional_light(direction=c(0.1,0.6,-1)),
                lookfrom = c(0,0,-2), lookat=c(0,0,0))
}
```

---

scene_from_list *Scene From List*

Description

Fast generation of rayvertex scenes from a list of objects (much faster than calling add_shape() on each object individually to build the scene). This returns a ray_scene object that cdoes

Usage

```
scene_from_list(scene_list)
```

Arguments

scene_list	List containing rayvertex mesh objects.

Value

ray_scene containing mesh info.

Examples

```
if(run_documentation()) {
 #Build a scene out of cubes including 87 * 61 = 5307 objects
 scene = list()
 volcol = rainbow(103)
 counter = 1
 for(i in 1:nrow(volcano)) {
   for(j in 1:ncol(volcano)) {
     scene[[counter]] = cube_mesh(position = c(i,(volcano[i,j]-94),j),
                            material = material_list(diffuse = volcol[volcano[i,j]-92],
                                                     ambient = volcol[volcano[i,j]-92],
                                                          ambient_intensity = 0.2))
     counter = counter + 1
   }
 }
 #Quickly generate the
 new_scene = scene_from_list(scene)
 new_scene |>
   rotate_mesh(c(0,10,0), pivot_point = c(44,0,31)) |>
   add_shape(xz_rect_mesh(position=c(44,0,31),scale=500,
                          material = material_list(diffuse="lightblue",
                                                   ambient = "lightblue",
                                                   ambient_intensity = 0.2))) |>
   rasterize_scene(lookfrom=c(500,500,500), lookat = c(44.00, 40.50, 31.00),
                   width=800,height=800, fov=0, ortho_dimensions = c(140,140),
                   light_info = directional_light(c(-0.6,1,0.6)))
}
```

`segment_mesh` *Segment 3D Model*

Description

Segment 3D Model

Usage

```
segment_mesh(
  start = c(0, -1, 0),
  end = c(0, 1, 0),
  radius = 0.5,
  direction = NA,
  from_center = TRUE,
  square = FALSE,
  material = material_list()
)
```

Arguments

start	Default c(0, 0, 0). Base of the segment, specifying x, y, z.
end	Default c(0, 1, 0). End of the segment, specifying x, y, z.
radius	Default 0.5. Radius of the cylinder.
direction	Default NA. Alternative to start and end, specify the direction (via a length-3 vector) of the arrow. Arrow will be centered at start, and the length will be determined by the magnitude of the direction vector.
from_center	Default TRUE. If orientation specified via direction, setting this argument to FALSE will make start specify the bottom of the cone, instead of the middle.
square	Default FALSE. If TRUE, will use a square instead of a circle for the cylinder.
material	Default material_list() (default values). Specify the material of the object.

Value

List describing the mesh.

Examples

```
if(run_documentation()) {
#Generate a segment in the cornell box.
generate_cornell_mesh() |>
  add_shape(segment_mesh(start = c(100, 100, 100), end = c(455, 455, 455), radius = 50)) |>
  rasterize_scene(light_info = directional_light(c(0,0.5,-1)))
}
if(run_documentation()) {
# Draw a line graph representing a normal distribution, but with metal:
xvals = seq(-3, 3, length.out = 30)
yvals = dnorm(xvals)

scene_list = list()
for(i in 1:(length(xvals) - 1)) {
  scene_list = add_shape(scene_list,
                     segment_mesh(start = c(555/2 + xvals[i] * 80, yvals[i] * 800, 555/2),
                           end = c(555/2 + xvals[i + 1] * 80, yvals[i + 1] * 800, 555/2),
                            radius = 10,
                            material = material_list(diffuse="purple", type="phong")))
}

generate_cornell_mesh() |>
  add_shape(scene_list) |>
  rasterize_scene(light_info = directional_light(c(0,0.5,-1)))
}
if(run_documentation()) {
#Draw the outline of a cube:

cube_outline = segment_mesh(start = c(100, 100, 100), end = c(100, 100, 455), radius = 10) |>
  add_shape(segment_mesh(start = c(100, 100, 100), end = c(100, 455, 100), radius = 10)) |>
  add_shape(segment_mesh(start = c(100, 100, 100), end = c(455, 100, 100), radius = 10)) |>
  add_shape(segment_mesh(start = c(100, 100, 455), end = c(100, 455, 455), radius = 10)) |>
```

```
  add_shape(segment_mesh(start = c(100, 100, 455), end = c(455, 100, 455), radius = 10)) |>
  add_shape(segment_mesh(start = c(100, 455, 455), end = c(100, 455, 100), radius = 10)) |>
  add_shape(segment_mesh(start = c(100, 455, 455), end = c(455, 455, 455), radius = 10)) |>
  add_shape(segment_mesh(start = c(455, 455, 100), end = c(455, 100, 100), radius = 10)) |>
  add_shape(segment_mesh(start = c(455, 455, 100), end = c(455, 455, 455), radius = 10)) |>
  add_shape(segment_mesh(start = c(455, 100, 100), end = c(455, 100, 455), radius = 10)) |>
  add_shape(segment_mesh(start = c(455, 100, 455), end = c(455, 455, 455), radius = 10)) |>
   add_shape(segment_mesh(start = c(100, 455, 100), end = c(455, 455, 100), radius = 10))

generate_cornell_mesh() |>
  add_shape(set_material(cube_outline,diffuse="dodgerblue",type="phong")) |>
  rasterize_scene(light_info = directional_light(c(0,0.5,-1)))
  }
if(run_documentation()) {
#Shrink and rotate the cube
generate_cornell_mesh() |>
  add_shape(
    scale_mesh(rotate_mesh(set_material(cube_outline,diffuse="dodgerblue",type="phong"),
                angle=c(45,45,45), pivot_point=c(555/2,555/2,555/2)),0.5,
                center=c(555/2,555/2,555/2))) |>
  rasterize_scene(light_info = directional_light(c(0,0.5,-1)))
}
```

`set_material` *Set Material*

Description

Set the material(s) of the mesh.

Usage

```
set_material(
  mesh,
  material = NULL,
  id = NULL,
  diffuse = c(0.5, 0.5, 0.5),
  ambient = c(0, 0, 0),
  specular = c(1, 1, 1),
  transmittance = c(0, 0, 0),
  emission = c(0, 0, 0),
  shininess = 50,
  ior = 1,
  dissolve = 1,
  illum = 1,
  texture_location = "",
  normal_texture_location = "",
  bump_texture_location = "",
  specular_texture_location = "",
```

```
  ambient_texture_location = "",
  emissive_texture_location = "",
  diffuse_intensity = 1,
  bump_intensity = 1,
  specular_intensity = 1,
  emission_intensity = 1,
  ambient_intensity = 1,
  culling = "back",
  type = "diffuse",
  translucent = TRUE,
  toon_levels = 5,
  toon_outline_width = 0.05,
  toon_outline_color = "black",
  reflection_intensity = 0,
  reflection_sharpness = 0,
  two_sided = FALSE
)
```

Arguments

`mesh`	The target mesh.
`material`	Default `NULL`. You can pass the output of the `material_list()` function to specify the material, or use the following individual settings.
`id`	Default `1`. Either a number specifying the material to change, or a character vector matching the material name.
`diffuse`	Default `c(0.5,0.5,0.5)`. The diffuse color.
`ambient`	Default `c(0,0,0)`. The ambient color.
`specular`	Default `c(1,1,1)`. The specular color.
`transmittance`	Default `c(0,0,0)`. The transmittance.
`emission`	Default `c(0,0,0)`. The emissive color.
`shininess`	Default `50.0`. The shininess exponent.
`ior`	Default `1.0`. The index of refraction. If this is not equal to `1.0`, the material will be refractive.
`dissolve`	Default `1.0`. The transparency.
`illum`	Default `1.0`. The illumination.
`texture_location`	Default `""`. The diffuse texture location.
`normal_texture_location`	Default `""`. The normal texture location.
`bump_texture_location`	Default `""`. The bump texture location.
`specular_texture_location`	Default `""`. The specular texture location.
`ambient_texture_location`	Default `""`. The ambient texture location.

emissive_texture_location
: Default "". The emissive texture location.

diffuse_intensity
: Default 1. The diffuse intensity.

bump_intensity
: Default 1. The bump intensity.

specular_intensity
: Default 1. The specular intensity.

emission_intensity
: Default 1. The emission intensity.

ambient_intensity
: Default 1. The ambient intensity.

culling
: Default "back". The culling type. Options are back, front, and none.

type
: Default "diffuse". The shader type. Options include diffuse,phong,vertex, and color.

translucent
: Default TRUE. Whether light should transmit through a semi-transparent material.

toon_levels
: Default 5. Number of color breaks in the toon shader.

toon_outline_width
: Default 0.05. Expansion term for model to specify toon outline width. Note: setting this property via this function currently does not generate outlines. Specify it during object creation.

toon_outline_color
: Default black. Toon outline color. Note: setting this property via this function currently does not color outlines. Specify it during object creation.

reflection_intensity
: Default 0.0. Intensity of the reflection of the environment map, if present. This will be ignored if the material is refractive.

reflection_sharpness
: Default 1.0. Sharpness of the reflection, where lower values have blurrier reflections. Must be greater than zero and less than one.

two_sided
: Default NULL. Whether diffuse materials should be two sided (normal is taken as the absolute value of the dot product of the light direction and the normal).

Value

Shape with new material

Examples

```
if(run_documentation()) {
#Set the material of an object
generate_cornell_mesh() |>
 add_shape(set_material(sphere_mesh(position=c(400,555/2,555/2),radius=40),
                        diffuse="purple", type="phong")) |>
 add_shape(set_material(sphere_mesh(position=c(555/2,220,555/2),radius=40),
                        dissolve=0.2,culling="none",diffuse="red")) |>
```

```
add_shape(set_material(sphere_mesh(position=c(155,300,555/2),radius=60),
                       material = material_list(diffuse="gold", type="phong",
                                                ambient="gold", ambient_intensity=0.4))) |>
rasterize_scene(light_info=directional_light(direction=c(0.1,0.6,-1)))
}
```

`smooth_normals_mesh` *Calculate Smooth Mesh Normals*

Description

Calculate Smooth Mesh Normals

Usage

```
smooth_normals_mesh(mesh, id = NA)
```

Arguments

`mesh`	The mesh.
`id`	Default `NA` (all shapes). The shape index to have new normals calculated.

Value

Mesh with new vertex normals

Examples

```
if(run_documentation()) {
 #Let's construct a mesh from the volcano dataset
 #Build the vertex matrix
  vertex_list = list()
  counter = 1
  for(i in 1:nrow(volcano)) {
    for(j in 1:ncol(volcano)) {
      vertex_list[[counter]] = matrix(c(j,volcano[i,j],i), ncol=3)
      counter = counter + 1
    }
  }
  vertices = do.call(rbind,vertex_list)

  #Build the index matrix
  index_list = list()
  counter = 0
  for(i in 1:(nrow(volcano)-1)) {
    for(j in 1:(ncol(volcano)-1)) {
      index_list[[counter+1]] = matrix(c(counter,counter+ncol(volcano),counter+1,
                                counter+ncol(volcano),counter+ncol(volcano)+1,counter + 1),
                                         nrow=2, ncol=3, byrow=TRUE)
      counter = counter + 1
```

```
    }
    counter = counter + 1
  }
  indices = do.call(rbind,index_list)
  #Construct the mesh
  volc_mesh = construct_mesh(vertices = vertices, indices = indices,
                             material = material_list(type="diffuse", diffuse="darkred",
                                                      ambient = "darkred", ambient_intensity=0.2))
  #Rasterize the no-normal scene
  scale_mesh(volc_mesh, scale = c(1,1/3,1)) |>
    center_mesh() |>
   rasterize_scene(lookfrom=c(-50,50,100),lookat=c(7,-15,0), fov=40,width=800,height=800,
                   light_info = directional_light(c(0,1,1)) |>
                     add_light(directional_light(c(1,1,-1))))

  #Smooth the mesh
  volc_mesh_smooth = smooth_normals_mesh(volc_mesh)

  #Rasterize the scene
  scale_mesh(volc_mesh_smooth, scale = c(1,1/3,1)) |>
    center_mesh() |>
   rasterize_scene(lookfrom=c(-50,50,100),lookat=c(7,-15,0), fov=40,width=800,height=800,
                   light_info = directional_light(c(0,1,1)) |>
                     add_light(directional_light(c(1,1,-1))))
}
```

`sphere_mesh` *Sphere 3D Model*

Description

Sphere 3D Model

Usage

```
sphere_mesh(
  position = c(0, 0, 0),
  scale = c(1, 1, 1),
  angle = c(0, 0, 0),
  pivot_point = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  radius = 1,
  low_poly = FALSE,
  normals = TRUE,
  material = material_list()
)
```

Arguments

position	Default c(0,0,0). Position of the mesh.
scale	Default c(1,1,1). Scale of the mesh. Can also be a single numeric value scaling all axes uniformly.
angle	Default c(0,0,0). Angle to rotate the mesh.
pivot_point	Default c(0,0,0). Point around which to rotate the mesh.
order_rotation	Default c(1,2,3). Order to rotate the axes.
radius	Default 1. Radius of the sphere.
low_poly	Default FALSE. If TRUE, will use a low-poly sphere.
normals	Default TRUE. Whether to include vertex normals.
material	Default material_list() (default values). Specify the material of the object.

Value

List describing the mesh.

Examples

```
if(run_documentation()) {
#Generate a sphere in the Cornell box.
generate_cornell_mesh() |>
  add_shape(sphere_mesh(position = c(555/2, 555/2, 555/2), radius = 100)) |>
  rasterize_scene(light_info = directional_light(c(0,0.5,-1)))
}
if(run_documentation()) {
#Generate a shiny sphere in the Cornell box
generate_cornell_mesh() |>
  add_shape(sphere_mesh(position = c(555/2, 100, 555/2), radius = 100,
                    material = material_list(diffuse = "gold",type="phong"))) |>
  rasterize_scene(light_info = directional_light(c(0.5,0.5,-1)))
}
if(run_documentation()) {
#Generate an ellipsoid in the Cornell box
generate_cornell_mesh() |>
  add_shape(sphere_mesh(position = c(555/2, 210, 555/2), radius = 100,
                        angle=c(0,30,0), scale = c(0.5,2,0.5),
                       material = material_list(diffuse = "dodgerblue",type="phong"))) |>
  rasterize_scene(light_info = directional_light(c(0.5,0.5,-1)))
}
```

---

subdivide_mesh *Subdivide Mesh*

Description

Applies Loop subdivision to the scene (or selected meshes).

Usage

```
subdivide_mesh(
  scene,
  id = NA,
  subdivision_levels = 2,
  simple = FALSE,
  normals = TRUE,
  verbose = FALSE
)
```

Arguments

scene	The scene to subdivide.
id	Default NA, all shapes. The index of which shape to subdivide.
subdivision_levels	Default 1. Number of Loop subdivisions to be applied to the mesh.
simple	Default FALSE. Whether to use simple subdivision, which does not change the appearance of the mesh but does create a finer mesh.
normals	Default TRUE. Whether to calculate subdivided vertex normals.
verbose	Default FALSE.

Value

Scene with shape added.

Examples

```
if(run_documentation()) {
#Subdivide the included R mesh
obj_mesh(r_obj(),position=c(-0.5,0,0)) |>
  add_shape(subdivide_mesh(obj_mesh(r_obj(),position=c(0.5,0,0)),
                           subdivision_levels = 2)) |>
  rasterize_scene(light_info = directional_light(direction=c(0.2,0.5,1)),fov=13)
}
```

---

swap_yz *Swap Y/Z Axis*

Description

Swap Y/Z Axis

Usage

```
swap_yz(mesh)
```

Arguments

mesh	A raymesh object.

Value

Mesh with Y and Z axis exchanged

Examples

```
# Flip a mesh that's originally aligned along the y-axis
if(run_documentation()) {
cyl_mat = material_list(ambient="red", ambient_intensity=0.3,
                        diffuse="red", diffuse_intensity=0.7)
change_material(cylinder_mesh(length = 3, position=c(0,2,0), material = cyl_mat),
                diffuse="green", ambient="green") |>
  add_shape(swap_yz(cylinder_mesh(position=c(0,2,0), length=3, material = cyl_mat))) |>
  rasterize_scene(lookfrom=c(10,10,10), lookat=c(0,0,0), fov=40,
                  light_info = directional_light(c(1,1,-1)),
                  line_info = generate_line(end=c(10,0,0)) |>
                  add_lines(generate_line(end=c(0,10,0),color="green")) |>
                  add_lines(generate_line(end=c(0,0,10),color="red")))
}
```

---

text3d_mesh *Text Object*

Description

Text Object

Usage

```
text3d_mesh(
  label,
  position = c(0, 0, 0),
  text_height = 1,
  orientation = "xy",
  font_color = "black",
  font_size = 100,
  font = "sans",
  font_lineheight = 12,
  background_color = "white",
  background_alpha = 0,
  angle = c(0, 0, 0),
  pivot_point = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  scale = c(1, 1, 1)
)
```

Arguments

label	Text string.
position	Default c(0,0,0). Position of the mesh.
text_height	Default 1. Height of the text.
orientation	Default xy. Orientation of the plane. Other options are yz and xz.
font_color	Default "black". The font color.
font_size	Default 100. The size of the font. Note that this does not control the size of the text, just the resolution as rendered in the texture.
font	Default "sans". A character string specifying the font family (e.g., "Arial", "Times", "Helvetica").
font_lineheight	Default 12. The lineheight for strings with newlines.
background_color	Default "white". The background color.
background_alpha	Default 0. The background opacity. 1 is fully opaque.
angle	Default c(0,0,0). Angle to rotate the mesh.
pivot_point	Default c(0,0,0). Point around which to rotate the mesh.
order_rotation	Default c(1,2,3). Order to rotate the axes.
scale	Default c(1,1,1). Scale of the mesh. Can also be a single numeric value scaling all axes uniformly.

Value

List describing the mesh.

Examples

```
if(run_documentation()) {
#Generate a label in the Cornell box.
generate_cornell_mesh() |>
 add_shape(text3d_mesh(label="Cornell Box", position=c(555/2,555/2,555/2),angle=c(0,180,0),
  text_height=120)) |>
  rasterize_scene(light_info = directional_light(c(0.1,0.4,-1)))
}
if(run_documentation()) {
#Change the orientation
generate_cornell_mesh() |>
  add_shape(text3d_mesh(label="YZ Plane", position=c(540,555/2,555/2),text_height=180,
                    orientation = "yz",angle=c(0,180,0))) |>
  add_shape(text3d_mesh(label="XY Plane", position=c(555/2,555/2,540),text_height=180,
                    orientation = "xy", angle=c(0,180,0))) |>
  add_shape(text3d_mesh(label="XZ Plane", position=c(555/2,15,555/2),text_height=180,
                    orientation = "xz", angle=c(0,180,0))) |>
  rasterize_scene(light_info = directional_light(c(0.1,0.4,-1)))
  }
```

```
if(run_documentation()) {
#Add an label in front of a sphere and change the font
generate_cornell_mesh() |>
 add_shape(text3d_mesh(label="Cornell Box", position=c(555/2,555/2,555/2),text_height=180,
                       font = "Serif", font_color="orange",
                       angle=c(0,180,0))) |>
  add_shape(text3d_mesh(label="Sphere", position=c(555/2,130,100),text_height=100,
                       font = "sans",
                       font_color="lightblue",angle=c(0,180,40))) |>
  add_shape(sphere_mesh(radius=100,position=c(555/2,100,555/2),
                       material=material_list(diffuse="purple",type="phong"))) |>
  rasterize_scene(light_info = directional_light(c(0.1,0.4,-1)))
  }
if(run_documentation()) {
#A room full of b's
set.seed(1)
bee_scene = list()
for(i in 1:100) {
bee_scene = add_shape(bee_scene, text3d_mesh("B", position=c(20+runif(3)*525),
                                             font_color="yellow", text_height = 100,
                                             angle=c(0,180,0)))
}
generate_cornell_mesh() |>
  add_shape(bee_scene) |>
  rasterize_scene(light=directional_light(c(0,1,-1)))
}

if(run_documentation()) {
#A room full of bees
bee_scene = list()
set.seed(1)
for(i in 1:100) {
  bee_scene = add_shape(bee_scene, text3d_mesh("\U1F41D", position=c(20+runif(3)*525),
                                               font_color="yellow", text_height = 100,
                                               angle=c(0,180,0)))
}
generate_cornell_mesh() |>
  add_shape(bee_scene) |>
  rasterize_scene(light=directional_light(c(0,1,-1)))
}
```

`torus_mesh` *Torus 3D Model*

Description

Torus 3D Model

Usage

```
torus_mesh(
  position = c(0, 0, 0),
  scale = c(1, 1, 1),
  angle = c(0, 0, 0),
  pivot_point = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  radius = 0.5,
  ring_radius = 0.2,
  sides = 36,
  rings = 36,
  material = material_list()
)
```

Arguments

position	Default `c(0,0,0)`. Position of the mesh.
scale	Default `c(1,1,1)`. Scale of the mesh. Can also be a single numeric value scaling all axes uniformly.
angle	Default `c(0,0,0)`. Angle to rotate the mesh.
pivot_point	Default `c(0,0,0)`. Point around which to rotate the mesh.
order_rotation	Default `c(1,2,3)`. Order to rotate the axes.
radius	Default `0.5`. The radius of the torus.
ring_radius	Default `0.2`. The radius of the ring.
sides	Default `36`. The number of faces around the ring when triangulating the torus.
rings	Default `36`. The number of faces around the torus.
material	Default `material_list()` (default values). Specify the material of the object.

Value

List describing the mesh.

Examples

```
if(run_documentation()) {
#Plot a group of tori in the cornell box
generate_cornell_mesh(ceiling = FALSE) |>
 add_shape(torus_mesh(position=c(555/2,555/3,555/2), angle=c(20,0,45),
                      radius=120, ring_radius = 40,
                      material = material_list(diffuse="dodgerblue4",type="phong",
                                               ambient="dodgerblue4",ambient_intensity=0.2))) |>
 add_shape(torus_mesh(position=c(400,400,555/2), angle=c(20,200,45),radius=80, ring_radius = 30,
                      material=material_list(diffuse="orange",type="phong",
                                             ambient="orange",ambient_intensity=0.2))) |>
 add_shape(torus_mesh(position=c(150,450,555/2), angle=c(60,180,0),radius=40, ring_radius = 20,
                      material=material_list(diffuse="red",type="phong"))) |>
 rasterize_scene(light_info = directional_light(c(0,1,-2)))
}
```

translate_lines *Translate Lines*

Description

Translate Lines

Usage

```
translate_lines(lines, position = 1)
```

Arguments

lines	The line scene.
position	Default c(0,0,0). The translation vector.

Value

Translated line matrix.

Examples

```
if(run_documentation()) {
#Generate a cube out of lines
cube_outline = generate_line(start = c(-1, -1, -1), end = c(-1, -1, 1)) |>
  add_lines(generate_line(start = c(-1, -1, -1), end = c(-1, 1, -1))) |>
  add_lines(generate_line(start = c(-1, -1, -1), end = c(1, -1, -1))) |>
  add_lines(generate_line(start = c(-1, -1, 1), end = c(-1, 1, 1))) |>
  add_lines(generate_line(start = c(-1, -1, 1), end = c(1, -1, 1))) |>
  add_lines(generate_line(start = c(-1, 1, 1), end = c(-1, 1, -1))) |>
  add_lines(generate_line(start = c(-1, 1, 1), end = c(1, 1, 1))) |>
  add_lines(generate_line(start = c(1, 1, -1), end = c(1, -1, -1))) |>
  add_lines(generate_line(start = c(1, 1, -1), end = c(1, 1, 1))) |>
  add_lines(generate_line(start = c(1, -1, -1), end = c(1, -1, 1))) |>
  add_lines(generate_line(start = c(1, -1, 1), end = c(1, 1, 1))) |>
  add_lines(generate_line(start = c(-1, 1, -1), end = c(1, 1, -1)))
rasterize_lines(cube_outline,fov=40,lookfrom=c(1,2,10),lookat=c(0,0,0))
}
if(run_documentation()) {
#Scale the cube uniformly
translated_cube = color_lines(translate_lines(cube_outline,c(1,1,1)),"red")
translated_cube2 = color_lines(translate_lines(cube_outline,c(-1,-1,-1)),"green")

cube_outline |>
  add_lines(translated_cube) |>
  add_lines(translated_cube2) |>
  rasterize_lines(fov=40,lookfrom=c(1,2,10),lookat=c(0,0,0))
}
```

translate_mesh *Translate Mesh*

Description

Translate Mesh

Usage

```
translate_mesh(mesh, position = c(0, 0, 0))
```

Arguments

mesh	The mesh.
position	Default `c(0,0,0)`. The translation vector.

Value

Translated mesh

Examples

```
if(run_documentation()) {
#Translate a mesh in the Cornell box
robj = obj_mesh(r_obj(), scale=150,angle=c(0,180,0))
generate_cornell_mesh() |>
 add_shape(translate_mesh(robj,c(400,100,155))) |>
 add_shape(translate_mesh(robj,c(555/2,200,555/2))) |>
 add_shape(translate_mesh(robj,c(155,300,400))) |>
 rasterize_scene(light_info=directional_light(direction=c(0.1,0.6,-1)))
}
```

validate_mesh *Validate Mesh Data*

Description

This function takes a mesh and validates it. The mesh should be a list with "shapes", "materials", "vertices", "texcoords", "normals", and "material_hashes" entries.

Usage

```
validate_mesh(mesh, validate_materials = TRUE)
```

Arguments

`mesh`	List. A mesh is a list as described above.
`validate_materials`	Default `TRUE`. Whether or not to validate "materials".

Value

A mesh.

Shapes

Each "shapes" entry should be a list with "mesh", "name", and "material" entries. Each "mesh" entry should have "indices", "tex_indices", "norm_indices", "material_ids", "has_vertex_tex", and "has_vertex_normals". The indices should not exceed the number of rows in their corresponding vertex/normal/texcoord data. There should be no NA/NaN values in the vertex/normal/texcoord data.

Materials (for rayvertex package only)

Each "materials" entry is expected to be a list with several entries with specific required lengths, as listed below:

Attribute	Length	Type
diffuse	3	Numeric
ambient	3	Numeric
specular	3	Numeric
transmittance	3	Numeric
emission	3	Numeric
shininess	1	Numeric
ior	1	Numeric
dissolve	1	Numeric
illum	1	Numeric
diffuse_texname	1	Character
normal_texname	1	Character
bump_texname	1	Character
specular_texname	1	Character
ambient_texname	1	Character
emissive_texname	1	Character
diffuse_intensity	1	Numeric
bump_intensity	1	Numeric
specular_intensity	1	Numeric
emission_intensity	1	Numeric
ambient_intensity	1	Numeric
culling	1	Character
type	1	Character
translucent	1	Logical
toon_levels	1	Numeric
toon_outline_width	1	Numeric
toon_outline_color	3	Numeric

reflection_intensity	1	Numeric
reflection_sharpness	1	Numeric
two_sided	1	Logical

Note: This materials validation only applies to the rayvertex package. Other renderers might choose to use their own information in the material list.

Examples

```
# validate a mesh
mesh = validate_mesh(sphere_mesh())
```

`write_scene_to_obj` *Write the scene to an OBJ file*

Description

Writes the current scene to a Wavefront OBJ file, with or without materials

Usage

```
write_scene_to_obj(scene, filename, materials = TRUE, fileext = ".obj")
```

Arguments

`scene`	A rayvertex scene.
`filename`	The filename for the OBJ file.
`materials`	Default `TRUE`. Whether to write an MTL file to specify the materials for the OBJ.
`fileext`	Default `".obj"`. The file extension to add to the filename.

Value

None

Examples

```
if(run_documentation()) {
tmpfile = tempfile(fileext = ".obj")
write_scene_to_obj(generate_cornell_mesh(), tmpfile)
}
```

xy_rect_mesh *XY Rectangle 3D Model*

Description

XY Rectangle 3D Model

Usage

```
xy_rect_mesh(
  position = c(0, 0, 0),
  scale = c(1, 1, 1),
  angle = c(0, 0, 0),
  pivot_point = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  material = material_list()
)
```

Arguments

position	Default `c(0,0,0)`. Position of the mesh.
scale	Default `c(1,1,1)`. Scale of the mesh. Can also be a single numeric value scaling all axes uniformly.
angle	Default `c(0,0,0)`. Angle to rotate the mesh.
pivot_point	Default `c(0,0,0)`. Point around which to rotate the mesh.
order_rotation	Default `c(1,2,3)`. Order to rotate the axes.
material	Default `material_list()` (default values). Specify the material of the object.

Value

List describing the mesh.

Examples

```
if(run_documentation()) {
generate_cornell_mesh() |>
  add_shape(xy_rect_mesh(position = c(555/2, 100, 555/2), scale=200,
             material = material_list(diffuse = "purple"),angle=c(0,180,0))) |>
  rasterize_scene(light_info = directional_light(c(0,0.5,-1)))
}
if(run_documentation()) {
#Rotate the plane and scale
generate_cornell_mesh() |>
 add_shape(xy_rect_mesh(position = c(555/2, 100, 555/2), scale=c(200,100,1), angle=c(0,180,0),
             material = material_list(diffuse = "purple"))) |>
  rasterize_scene(light_info = directional_light(c(0,0.5,-1)))
}
```

`xz_rect_mesh` *XZ Rectangle 3D Model*

Description

XZ Rectangle 3D Model

Usage

```
xz_rect_mesh(
  position = c(0, 0, 0),
  scale = c(1, 1, 1),
  angle = c(0, 0, 0),
  pivot_point = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  material = material_list()
)
```

Arguments

`position`	Default `c(0,0,0)`. Position of the mesh.
`scale`	Default `c(1,1,1)`. Scale of the mesh. Can also be a single numeric value scaling all axes uniformly.
`angle`	Default `c(0,0,0)`. Angle to rotate the mesh.
`pivot_point`	Default `c(0,0,0)`. Point around which to rotate the mesh.
`order_rotation`	Default `c(1,2,3)`. Order to rotate the axes.
`material`	Default `material_list()` (default values). Specify the material of the object.

Value

List describing the mesh.

Examples

```
if(run_documentation()) {
generate_cornell_mesh() |>
  add_shape(xz_rect_mesh(position = c(555/2, 100, 555/2), scale=200,
             material = material_list(diffuse = "purple"))) |>
  rasterize_scene(light_info = directional_light(c(0,0.5,-1)))
}
if(run_documentation()) {
#Rotate the plane and scale
generate_cornell_mesh() |>
 add_shape(xz_rect_mesh(position = c(555/2, 100, 555/2), scale=c(200,1,100), angle=c(0,30,0),
             material = material_list(diffuse = "purple"))) |>
  rasterize_scene(light_info = directional_light(c(0,0.5,-1)))
}
```

yz_rect_mesh *YZ Rectangle 3D Model*

Description

YZ Rectangle 3D Model

Usage

```
yz_rect_mesh(
  position = c(0, 0, 0),
  scale = c(1, 1, 1),
  angle = c(0, 0, 0),
  pivot_point = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  material = material_list()
)
```

Arguments

position	Default `c(0,0,0)`. Position of the mesh.
scale	Default `c(1,1,1)`. Scale of the mesh. Can also be a single numeric value scaling all axes uniformly.
angle	Default `c(0,0,0)`. Angle to rotate the mesh.
pivot_point	Default `c(0,0,0)`. Point around which to rotate the mesh.
order_rotation	Default `c(1,2,3)`. Order to rotate the axes.
material	Default `material_list()` (default values). Specify the material of the object.

Value

List describing the mesh.

Examples

```
if(run_documentation()) {
generate_cornell_mesh() |>
 add_shape(yz_rect_mesh(position = c(555/2, 100, 555/2), scale=c(200,1,200), angle=c(0,0,0),
            material = material_list(diffuse = "purple"))) |>
  rasterize_scene(light_info = directional_light(c(0,0.5,-1)))
}
if(run_documentation()) {
#Rotate and scale
generate_cornell_mesh() |>
 add_shape(yz_rect_mesh(position = c(555/2, 100, 555/2), scale=c(300,1,200), angle=c(0,45,0),
            material = material_list(diffuse = "purple"))) |>
  rasterize_scene(light_info = directional_light(c(0,0.5,-1)))
}
```

Index